UNIVERSITY OF OSLO
Department of Informatics

Value Matrix and Domain Map - Boundary Objects for Systemic Innovation

Benjamin Johansson

August 1, 2012

Abstract

Information overload is a complex and growing problem that many systems have tried to remedy. Recognizing that technology alone will probably not be enough to solve this problem, and that conventional knowledge work practices need to change to take advantage of existing or new tools, Knowledge Federation has self-organized as a community for doing systemic innovation. That is, they work to redesign and change the practices in key areas, such as public informing, education and science, of knowledge work.

Knowledge Federation is a transdisciplinary community, which consists of experts and stakeholders from a variety of fields, both technical and non-technical. Thus, the challenge is to provide the enabling technology in such a way that the technical details of the implementation are "encapsulated" or hidden, and that exactly those functions that are needed and natural for systemic innovation are "exported" or provided.

In this thesis, we address this challenge by introducing two initial prototypes for "boundary objects", objects that serve as communication channels between two domains. Here, the two domains are the technical domain of tool builders, such as Topic Maps, Semantic Web and various IBIS implementations, and the non-technical domain where systemic innovation takes place. A specific purpose of these objects is to enable the creation of a suitable "knowledge work ecology" where the right kind of practices are supported. That is, the ones that are needed to remedy the information overload.

The first object is the Domain Map Object (DMO), which can be likened to a filing cabinet, or a place for organizing and storing knowledge resources. It can also be viewed as a map, or a collection of maps, whose purpose is to show a high-level overview of the subject domain so that what is worth seeing can be easily located. In other words, the DMO provides affordances for organizing knowledge, which naturally stimulates the suitable practices.

Our other object is the Value Matrix Object (VMO), which is an object attached to every resource in a domain, accumulating all data that can be relevant for computing the value of the associated resource with respect to a given query or context. Our definition of "resource" includes users, specifically authors, in addition to knowledge resources. In particular, the VMO provides affordances for rewarding human users for right behavior, such as organizing knowledge resources and taking time to produce high-quality content instead of focusing on quantity, by keeping track of all contributions and their value. Thus, the VMO can be used by system builders to create an ecology that rewards both production of high quality knowledge as well as contribution to knowledge organization.

Besides describing the two objects, we design and implement a prototype that shows the objects' main capabilities. We complete the functionality of our objects as boundary objects by inviting people from the two relevant communities to test the prototype and answer a questionnaire. At the same time, this can be seen as an experiment to test the feasibility and usability of our objects. Based on the results of the tests, we give suggestions for improving the present boundary object prototypes.

Acknowledgments

First and foremost, I want to thank my main supervisor, Dino Karabeg, for his excellent support, guidance, ideas and invaluable input during the work on this thesis. I also want to thank all those who took their valuable time to participate in the evaluation of the prototype. Finally, I thank my fiancée, family and friends for their support and encouragement during the work with this thesis.

Benjamin Johansson University of Oslo August 1, 2012

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

> "As long as the centuries continue to unfold, the number of books will grow continually, and one can predict that a time will come when it will be almost as difficult to learn anything from books as from the direct study of the whole universe. It will be almost as convenient to search for some bit of truth concealed in nature as it will be to find it hidden away in an immense multitude of bound volumes."
>
> – Denis Diderot [15]

1.1 Information Overload - A Problem Both Historical and Acute

Already in 1755 Diderot predicted that the continual growth of information resources would eventually lead to the problem that we call *information overload* [15][4]. The term refers to a state where too much available information negatively affects a person's ability to understand an issue or make a decision [112]. Diderot is joined by Nietzsche in highlighting information overload as a problem, although Nietzsche focuses on the human reaction to an overload of impressions rather than the cause.

> *"Sensibility immensely more irritable (dressed up moralistically: the increase in pity); the abundance of disparate impressions greater than ever: cosmopolitanism in foods, literatures, newspapers, forms, tastes, even landscapes. The tempo of this influx prestissimo; the impressions erase each other; one instinctively resists taking in anything, taking anything deeply, to 'digest' anything; a weakening of the power to digest results from this. A kind of adaptation to this flood of impressions takes place: men unlearn spontaneous action, they merely react to stimuli from outside. They spend their strength partly in assimilating things, partly in defense, partly in opposition. Profound weakening of spontaneity: the historian, critic, analyst, the interpreter, the observer, the collector, the reader – all of them reactive talents – all science!"* [79]

In the above quote, Nietzsche talks about how we react to the "abundance of impressions" we experience in our daily lives. He says that we defend ourselves from these impressions, thus reducing our ability to take in information and reflect upon it. Additionally, we become less receptive to new ideas and act only upon stimuli from our environment. If Nietzsche is right, then we are affected by large amount of information oppositely from what is normally intended and believed [57].

Naturally, information overload has become an even larger issue after the introduction of the Internet. At the moment, more than 2 billion people use the Internet regularly, having access to over 8 billion pages containing information [38][63]. In fact, each and every day, an average person observes the same amount of information that a knowledge worker was able to absorb in his entire lifetime 100 years ago [95]. According to Marshall McLuhan one of the effects of living with electric information is that we live habitually in a state of information overload [73]. This is obviously because digital technology allows us to overproduce information. In fact, in a study conducted by Varian and Lyman of UC Berkeley it was found that the amount of information increased by over 30% per year during the period of observation [70]. In other words, the amount of information doubles roughly every three years. Furthermore, computer processing power and memory increases over time, while the human mind is not getting any faster [46]. It acts as a bottleneck.

A side effect of the increasing amount of information on the Internet is that not all of it is reliable or accurate because there is no authority conducting quality control on the material. Additionally, many websites fail to include the information required for visitors to be able to verify the content on the site [30]. Thus, we run the risk of being mislead or misinformed while looking at information on the web, which in turn influences us to make wrong decisions.

There has been conducted some research on the effects of information overload and how to reduce the impact of it. It has been claimed that information overload could have an impact on the thought process by obstructing understanding, making learning more difficult as well as affecting deep thinking. Further, such overload could result in retaining only a small piece of the information presented [18]. Other cognitive scientists assert that information overload is better understood as organization underload. They suggest that the problem is not the amount of available information, but rather that we do not know how to organize it and use it effectively and with efficiency. One of these scientists is Edward Tufte who makes this point by saying: *"What about confusing clutter? Information overload? Doesn't data have to be "boiled down" and "simplified"? These common questions miss the point, for the quantity of detail is an issue completely separate from the difficulty of reading. Clutter and confusion are failures of design, not attributes of information."* [106]

1.2 Approaches to Cure and Our Contribution

Over the years, impressive technologies and ideas have been developed to remedy the information overload problem. Examples of such technologies include mind maps and concept maps, topic maps, semantic web and naturally traditional knowledge organization. An overview over these, and other systems, are presented and discussed in Chapter 2.

It is clear that existing systems contribute in one level or another to curing information overload. However, conventional knowledge work practices and values have not been changed to accommodate this powerful technology. Instead, the practices remain the same while new technologies are simply added to them. With conventional knowledge work practices, we mean the standard practices in, for example, academia and journalism where knowledge workers are evaluated by their volume production alone [55]. There is no notion that researchers and other knowledge workers should organize their own or others' publications, since there is no reward for doing so. Production of knowledge without having a way to methodically and coherently organize and evaluate it breeds overload, simply because more and more information is added to the already vast sea of knowledge. The right kind of practices for knowledge work include activities such as organizing knowledge resources and taking time to produce high-quality content, instead of focusing on mere quantity. These activities should be considered valuable for both academic communities and employers alike. Additionally, it should be positive for knowledge workers'

careers. Thus, new practices must suitably reward all kinds of knowledge work activities.

In order to be able to change the current knowledge work practices, systemic innovation must be performed. We aim to add the missing piece of the puzzle by complementing existing technology and bring it to use, by changing the practices and performing systemic innovation. For example, Topic Maps provide powerful functionality, but it not widely used in day-to-day knowledge work. This can be changed, by enabling systemic innovation.

A community known as Knowledge Federation has acknowledged that technology alone is sufficient to solve information overload, and that conventional knowledge work practices need to change to take advantage of existing or new tools. Thus, Knowledge Federation has self-organized to tackle the task of systemic innovation, including real-life systemic change, as well as designing sociotechnical systems [74][58]. The community consists of creative knowledge workers from a variety of fields, such as academic researchers, journalists, IT developers and others. We can view the members of the organization as parts of a collective mind that are capable of enabling systemic improvement by providing reliable shared insight [58]. The fact that the Knowledge Federation members originate from different background help them develop sociotechnical systems. This is because no single community of people with the same background have the expertise or the authority to develop such systems, like changing conventional knowledge work practices, on a large scale [58]. Instead, people with different areas of expertise must collaborate, just like what the Knowledge Federation is doing. Douglas Engelbart supports this view. He saw the best approach towards handling increasingly complex problems is to apply our collective creativity in order to improve our collective creativity [66].

However, since Knowledge Federation members come from both technical and non-technical backgrounds, the challenge is to provide the enabling technology in such a way that the technical details of the implementation are "encapsulated" or hidden, and that exactly those functions that are needed and natural for systemic innovation are "exported" or provided.

Boundary Objects for Systemic Innovation

With this thesis, our goal is to help the Knowledge Federation community with systemic innovation of knowledge work. We contribute by introducing two boundary objects that we call the Domain Map Object and the Value Matrix Object. The boundary object concept was coined by Star and Griesemer in 1989 and refers to objects that serve as an interface between different communities of practice [100]. Boundary objects are not only shared by the various communities, but also viewed or used differently by each of them. We will now explain the purposes of our two objects.

The Domain Map Object (DMO) is a knowledge management object that is assigned to a community of interest. It is an online representation of the community's domain of interest and provides the means to access and organize knowledge resources in a subject-centric way, while it aims to minimize both overload and visual load [59]. In addition, a domain map offers the possibility of having multiple views of the domain it represents, for example showing various levels of detail or highlighting areas where knowledge is lacking. Members in a community may use the DMO to publish their papers, organize existing information in domain or locate information. Essentially, the Domain Map Object may be likened to Topic Maps with some additional functionality. However, instead of being a topical index into a domain it is rather a visual representation of a domain [55]. Overall, the purpose of the DMO is to provide affordances for organizing knowledge in a way that stimulates suitable practices.

The Value Matrix Object (VMO) is an object that is associated with a resource in the domain represented by a DMO. A resource can either be a document containing information or a person that is a member of the community. Thus, the VMO plays two similar, yet distinct roles. When attached to a resource, the VMO accumulates all data that could be relevant

for computing the value of that resource given a query. This data is accumulated through, for example, ratings provided by members of the community about the quality or importance of the resource. The other role is when attached to a person. Then, the VMO collects all information about the member's contribution, which is done so that undervalued contributions like organization and evaluation of knowledge is suitably rewarded [59][55]. By providing a reward system, right behavior is naturally stimulated.

Above we described how our two objects function in use as a system in a community of interest. However, they also function as boundary objects, which help us achieve the goal of enabling systemic innovation. As boundary objects, our two objects fulfill two purposes. For the first purpose, the boundary objects act, on a meta-level, as a communication device that lets different communities communicate with one another to federate the collective knowledge and insights. Consider the following two communities: System builders and tool makers. A boundary object lets the tool makers give the right tools to system builders by encapsulating tools, for example topic maps, and export the kind of functions that the system builders need to create new sociotechnical systems. The communication works the other way as well. The system builders can give tool makers a set of patterns that they want the tool makers to develop technology for [59].

The other purpose our boundary objects fulfills is, as tools, to enable the creation of a knowledge work ecology, which is possible due to the affordances provided by our two objects. While the DMO encourages the act of organizing and publishing knowledge in a coherent system, the VMO makes sure that both high quality knowledge is brought forward and that workers are suitably rewarded for their contributions. In other words, the two objects support the right practices.

In the final system the end users never see our two building blocks. We wish to reduce complexity so that the system feels natural to use. Thus, it is important that the objects *hide implementation and export function*, so that the details are not important and the application is easy to use for the end user. Liken this philosophy to how a car functions. It "exports" the wheel and the pedals, while the details of the engine is "hidden" [59]. The two boundary objects, DMO and VMO, will be discussed more thoroughly in Chapter 3 and Chapter 4, respectively.

1.3 Outline

The rest of this thesis is organized in the following manner: Chapter 2 will go through a set of existing technology that has provided some of the ground work and inspiration for the Value Matrix Object and the Domain Map Object, as well as provide an introduction to reputation systems which are particularly relevant in relation to the VMO. In Chapter 3, we present the Domain Map Object and its core concepts, while we in Chapter 4 take a deeper look at the Value Matrix Object and how it is used to federate resource value. Our prototype design is explained in Chapter 5, while in-depth technical implementation details are provided in Chapter 6. Chapter 7 deals with the evaluation of the prototype design. Finally, in Chapter 8, we sum up our work and present what can be done in the future.

This thesis also contains three appendices. Appendix A contains access to the client side source code of the prototype, as well as a short explanation of each class and how new builds of the prototype can be compiled. In Appendix B, we give access to the server side files and explain how the database can be accessed. The final appendix, Appendix C, presents all the answers we received from our evaluation questionnaires as well as a transcript of the evaluation invitation email we sent to all participants.

Chapter 2

Related Systems and Technology

> "There is only one way in which a person acquires a new idea: by the combination or association of two or more ideas he already has into a new juxtaposition in such a manner as to discover a relationship among them of which he was not previously aware."
>
> – Francis A. Cartier

In this chapter, we will present existing systems, technologies and techniques that are used for knowledge organization. Naturally, knowledge organization is the art of organizing data in a way so that it will be easier to locate a specific piece of information. We will not present a complete overview over such systems, but rather examples that have influenced our work, specifically the Domain Map Object. Then, we will discuss reputation systems, as we use concepts from them in the design of the Value Matrix Object, specifically in the computation of resource value.

2.1 Visual Organization Techniques

Tufte, who specializes in the presentation of informational graphics, such as charts and diagrams, and data visualization, does not believe in information overload. He says: *"Overload, clutter, and confusion are not attributes of information, they are failures of design. So if something is cluttered, fix your design, don't throw out information. If something is confusing, don't blame your victim – the audience – instead, fix the design."* [12]. What he essentially is saying is that having a great visual design eliminates the overload. Even if Tufte is talking about overload in the context of presentation of data, we can still use his idea that the solution lies in the design when talking about information overload in the context of amount of information.

Research has shown that graphical representation of information has cognitive advantages over textual representation of information. One of the main claims is that written information is often presented in ways that are difficult to follow, understand or remember [14][48][44]. Furthermore, Card, Mackinlay and Shneiderman have conducted experiments using visualization techniques and interactive graphics which they claim stimulate the mind and amplify the thought process in humans [11]. In other words, the idea behind visual knowledge organization software is that by applying visual tools and graphics, people will have an easier time learning and remembering as well as being able to locate the information they are looking for.

2.1.1 Mind Maps

One way to organize information visually is to use mind maps. These maps are generally used as an aid to studying and organizing information as well as solving problems and making decisions. Additionally, a mind map can help us sort out and structure complicated ideas. A mind map consists of a single word, idea or concept, in the middle, with words, ideas and concepts that can be associated with the main word, around it. These elements can be grouped in order to represent a connection between them [10]. Figure 2.1 shows the general structure of a mind map.

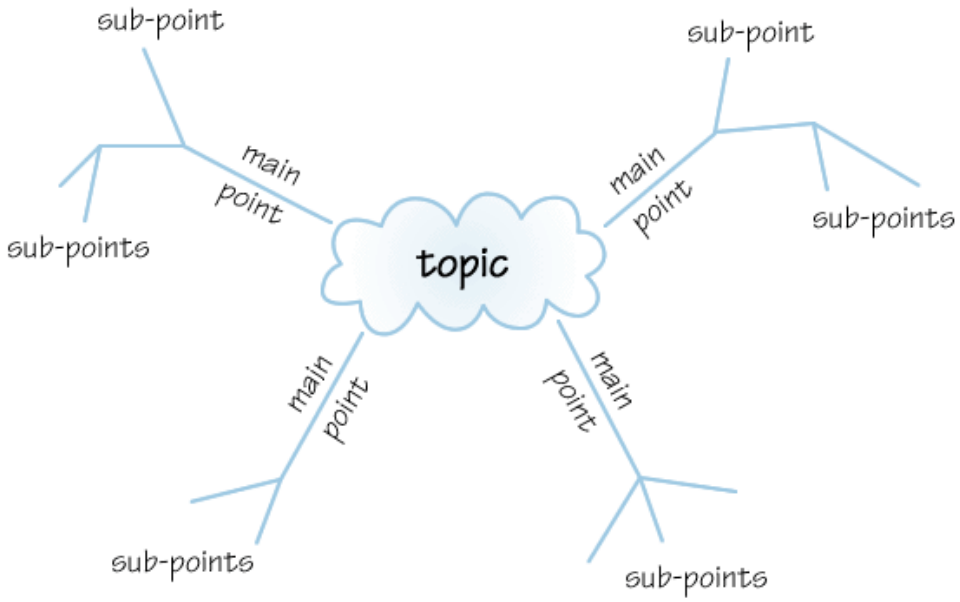

Figure 2.1: Structure of a mind map

Tony Buzan, who claims to have invented modern mind mapping, says that using a mind map for taking notes is superior to other methods because we use our brain actively and thus do not fall into a "semi-hypnotic trance" [10]. With regard to learning and remembering, a study conducted by Farrand et al. concluded that using mind maps is an effective study technique, but that learners preferred other methods because mind mapping was an unfamiliar technique [29].

In relation to information overload, mind maps can help us organize information and stay focused. For example, we can use a mind map as a part of writing knowledge resources by organizing our background material. Additionally, mind maps can be used to design and structure the knowledge resources we create, enhancing our work and indirectly reducing overload.

2.1.2 Concept Maps

Concept maps are similar to mind maps in that they are both graphical tools for organizing and representing knowledge as well as representing relationships between ideas. The main difference between the two is where a mind map is spontaneous and informal, a concept map is more structure and less pictorial in nature [22]. Additionally, their goals are different. A mind map is used to create spontaneous associative elements, while a concept map is created to outline relationships between ideas [22].

The concept map consists of concepts that are enclosed in circles or boxes, and lines between concepts that indicates that there is a relationship between them. In addition, there can be words on these lines that specifies the relationship between the two concepts a line links together. Thus, a concept map is a diagram showing the relationships among concepts in a downward-branching hierarchical structure with the general concepts at the top of the map and the more specific, less general concepts arranged hierarchically below [80]. Figure 2.2 shows an example of a concept map.

Concepts maps also have *propositions*, which are statements about objects or events in the map domain. Propositions are created by having two or more concepts connected by lines that form a meaningful statement [80]. This attribute makes it possible for concept maps to be used in ontology-building, which of course is an important element in the semantic web [47].

It has been shown that concept mapping helps learners learn, researchers create new knowledge, administrators to better structure and manage organizations, writers to write, and evaluators assess learning [80].

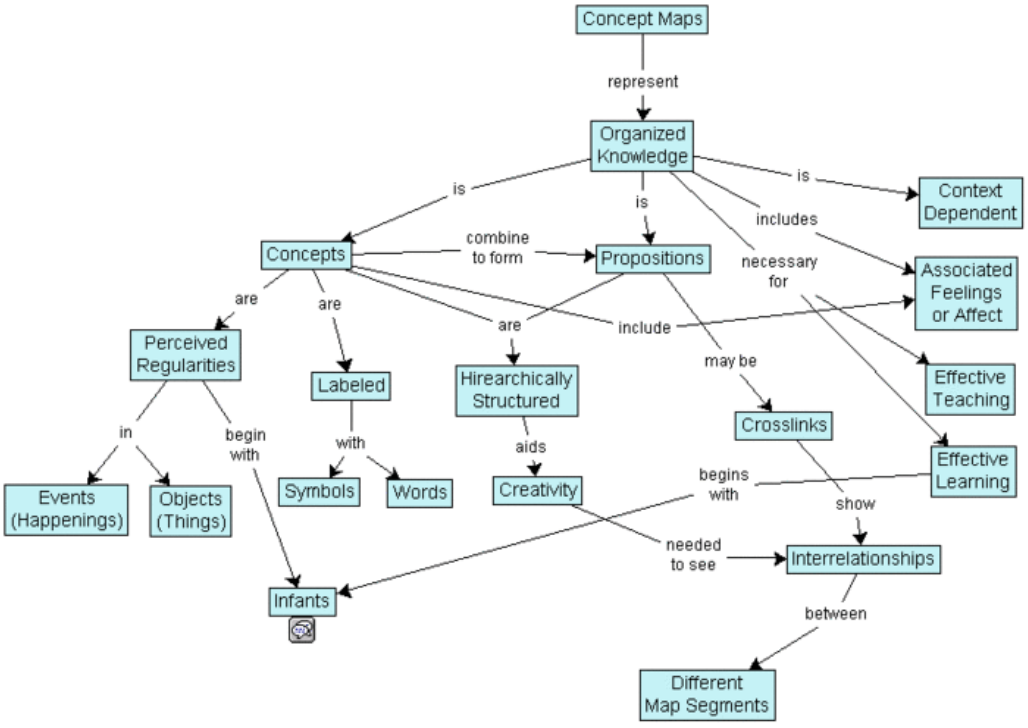

Figure 2.2: An example of a concept map

2.1.3 Category Maps

Category maps are very different from both mind maps and concept maps. While both concept maps and mind maps focus on relatively small and restricted domains, category maps are a two-dimensional graphical tool for Internet browsing that are based on Kohonen's self-organizing map [112][62]. In fact, experiments have shown that category maps are capable of categorizing large Internet information spaces [16].

What makes these maps suitable for Internet browsing is that the self-organizing map algorithm automatically categorizes a large information space into smaller, more manageable sub-spaces. In addition, these sub-spaces are transformed into a two-dimensional graphical representation that users can navigate and locate areas of interest. The sub-spaces, or categories, are generated based on the semantics of the documents in the domain, for example the Internet, that the self-organizing map algorithm processes [112].

Visual load is an obvious problem with category maps since as the information space grows, more categories emerge and are added to the map [3]. Soon, there are so many categories packed closely together on the map that it is impossible to see the details. Thus, users will have trouble finding areas of interest [112]. There has been presented solutions to this problem, like being able to zoom into parts of the map in order to see the details. However, by zooming, in the overall structure of the category map is lost [112][111].

Yang et al. have proposed and conducted experiments on two view techniques supposed to battle the visual load problem. These two views are the fisheye view and the fractal view. The fisheye view magnifies the area of focus and shrinks the surrounding areas, while the fractal view lets users control the amount of information displayed by adjusting the fractal value that filters information that is not relevant to the area of interest. This view does not maintain the global map structure [112][111].

In their experiments, Yang et al. found that both views significantly improve the effectiveness of visualization, but that the fractal view performed considerable better than the fisheye view [112].

2.2 Traditional Knowledge Organization

Naturally, visual organization is not the only way to organize information. Knowledge Organization (KO) is a field of study that is concerned with the nature and quality of knowledge organization processes (KOP) and knowledge organization systems (KOS) [41]. KOP is about activities such as indexing and classification performed in libraries, bibliographical databases and document description, while KOS is more concerned about the schemes used to organize and represent documents [41][114].

2.2.1 Knowlege Organization Approaches

There are many theoretical approaches to knowledge organization. The first approach is the traditional approach, which includes the Dewey Decimal Classification (DDC) and can be traced back to about 1876 [41]. DDC was developed by Melvil Dewey, whose interest was to develop a standardized and efficient way to manage library collections. Dewey's system categorizes books on library shelves in an efficient, specific and repeatable order that makes it easy to find any book and return it to its proper place on the library shelves [41]. This is done by associating each book into one of ten main classes. Each of these ten classes are divided into ten divisions, which each again has ten sections. The total is then ten main classes, 100 divisions and 1000 sections in Dewey's system [25]. Thus, each book is given a three-digit number, where the first digit represents the main class, the second digit indicates division and the third digit indicates section. For example, works about algebra & number theory are given the number 512, where the '5' represents the science class and the '1' represents the mathematics section. Furthermore, a decimal point may follow the third digit in order to achieve more specific degrees of classification.

Even after 100 years of research and development, the traditional approach to knowledge organization is still in a strong position. In fact, the DDC is the most widely used classification

system in the world and around 95% of school libraries and public libraries in the U.S. use it [81][99].

Another approach to knowledge organization is the facet-analytical approach, which is called the "modern classification theory". Its foundation can be traced back to 1933 and the publication of S.R. Ranganathan's *Colon Classification* [41]. Ranganathan proposed five common categories, or "facets", that subjects are broken down into. The subjects are, for example, book titles and the categories are Personality, Matter, Energy, Space and Time [41][86]. The categories are logical, a priori categories as they are not dynamically developed as new books are written.

Furthermore, each category, or facet, has its own classification or list of symbols. The idea is that a document is classified by taking one or more symbols from the appropriate facets and combining them according to certain rules [41][86]. This combination is meant to describe the subject matter of the classified document.

Ranganathan claims that the strength of a faceted system as opposed to an enumerative system like DDC, is that the faceted system supports the discovery of new knowledge while the enumarative system does not. This is the case, says Ranganathan, since new knowledge can be classified by the combination of a priori existing categories [41][87]. The faceted approach is used on the Web as well, for example in XFML which is an XML format for exchanging metadata between websites in the form of faceted hierarchies [108].

Other knowledge organization approaches include the information retrieval approach, the user oriented and cognitive views approach, bibliometric approaches and the domain analytic approach [42].

2.2.2 Knowledge Organization Systems

Hill defines the meaning of the term knowledge organization systems to encompass all types of schemes of organizing information and promoting knowledge management [43][114]. Examples of such schemes are classification schemes, taxonomies and ontologies. KOS aim to provide semantics, navigation and translation through labels, definitions and relationships in order to model the semantic structure of a domain [40]. Then, these systems can be used as services that facilitate both resource discovery and retrieval for humans and machines [114].

Knowledge organization systems can be split into four major groups, ranging simple to more complicated: Term lists, metadata-like models, classification and categorization and finally relationship models [43][40]. Term lists include pick lists, dictionaries, glossaries and synonym rings, while metadata-like models include authority files, directories and gazetteers. Classification and categorization systems include categorization schemes, classification schemes and taxonomies. Finally, relationship models include thesauri, semantic networks and ontologies.

Depending on the complexity of the KOS, they fulfill a set of fundamental function. These functions are eliminating ambiguity, controlling synonyms, establishing hierarchical and/or associative relationships and presenting properties [114]. In her article, Zeng presents various ways to fulfill these functions.

An increasing trend is that KOS structures are being integrated into web-based services and are no longer used only for indexing, organizing and searching, but also in learning and knowledge modeling. These networked KOS' do not only inherit properties from traditional KOS structures, but they also form new semantic structures that Zeng claims will have a greater impact then previously imagined [114].

2.2.3 Who Should do KO?

In his article "*Ten Long-Term Research Questions in Knowledge Organization*" Gnoli raises the question: Who should do KO? [36]. He says that traditionally it has been professionals trained

in using knowledge organization systems that have worked with the organization of knowledge. Further, he claims that this is no longer possible because the professionals cannot keep up with the explosive production of new documents.

One solution, Gnoli says, is that authors themselves provide metadata for the documents that they produce. This metadata should include the subject of the document, as well as some keywords and a description of the document content. Obviously, this is not an entirely new idea. Gnoli recognizes that authors of specialized papers have provided keywords with their own documents for decades. Additionally, it happens that authors in the same field of study review and index the work of their colleagues [36].

Finally, Gnoli presents a third possibility in that readers can perform knowledge organization instead of trained professionals or authors. This is possible because readers have access to collections of documents through digital networks and are able to add their own "tags" describing the documents. There are both supporters and critics of this practice. The supporters emphasize the democratic aspect, while the critics emphasize the lack of vocabulary control [36].

2.3 Examples of Knowledge Organization Systems

In this section, we will present examples of concrete online systems that use various techniques to organize knowledge and information. First, we discuss Topic Maps and Semantic Web as two non-visual systems, before we move on to visual systems such as Debategraph and SpicyNodes.

2.3.1 Topic Maps

As the Internet started to be a major source of knowledge documents, the need for a way to organize these documents resulted in the ISO standardized Topic Maps [6]. Topic Maps were made to increase the findability of information by creating a standard way to represent knowledge and make it interchangeable between maps. The initial purpose was to address the problem of too much information on the Internet and is the online equivalent of indexes in books [7]. Garshol claims that while there is a relationship between Topic Maps and traditional knowledge organization classification schemes, Topic Maps are not an extension of these schemes, but on a higher level entirely. He says: "*A summary of the relationship between topic maps and traditional classification schemes might be that topic maps are not so much an extension of the traditional schemes as on a higher level. That is, thesauri extend taxonomies, by adding more built-in relationships and properties. Topic maps do not add to a fixed vocabulary, but provide a more flexible model with an open vocabulary. A consequence of this is that topic maps can actually represent taxonomies, thesauri, faceted classification, synonym rings, and authority files, simply by using the fixed vocabularies of these classifications as a topic map vocabulary.*" [34]

A topic map consists of three main concepts; *topics*, *occurrences* and *associations*. Topics are the building blocks of a topic map. A topic is a multi-headed link and points to all occurrences of the topic. Each link accumulates every piece of information that is about a given subject. The subject of a topic is whatever the topic is about. In addition, topics can be grouped into *classes* or *types* such as *country*, *product* and *person*. Topics are grouped into these types in order to build specialized indexes, which improves search functionality. Topics are instantiated outside the information resources, and they collectively make a topic map [6].

The second concept, occurrences, relates to the information resources, i.e. the documents, that are relevant to a given topic. Occurrences for each topic can be divided into subgroups that are defined by a common role. For example, it is possible to separate a topic's occurrences by distinguishing graphics from text, main occurrences from ordinary occurrences and others. These subgroups are user-definable.[6]

Lastly, we have the associations concept. These are used to represent the relationship between topics. One use for topic associations is to let one topic be a container for other topics so that it possible to describe topic trees. With these trees, virtual table of contents can be built. Another use is to define an "employment" association between a person and a company in order to describe the relationship between these two topics. Topic associations are independent of the resource documents where the topic occurrences are found. Because of this, the associations represent a knowledge base which contains the essence of the information the designer of the Topic Map is creating.[6]

We can say that an instantiated topic map represents a structured view over a set of knowledge resources [7]. Alternatively, it is a topical index into a domain represented by the set of knowledge resources [55].

Figure 2.3 shows a high-level view of a topic map. In the figure, the red shapes represent the topics and their types, while the black lines point to each topic's occurrences. Lastly, the colored striped lines indicate that there is an association between topics. The same color means that the association is the same.

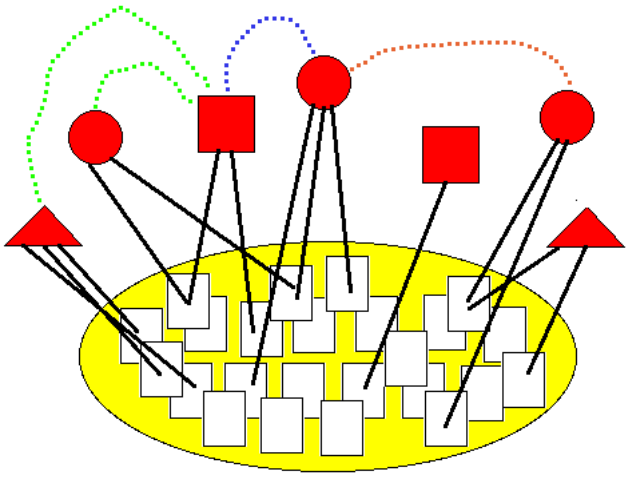

Figure 2.3: High-level view of a topic map

2.3.2 Semantic Web

Semantic Web has been defined as "a web of data that can be processed directly and indirectly by machines" by Tim Berners-Lee [5]. The idea behind the Semantic Web is to bring structure to the meaningful content of web pages, so that software agents have an environment where they can carry out sophisticated tasks for users. Today, the content on the Internet is designed for humans to read as a collection of documents, and not data and information that a computer can manipulate in a meaningful way. While computers can parse web pages and identifying links and such, they have no reliable way of processing the semantics of the page contents [5].

It is important to note that the Semantic Web is not a separate Web, but an extension of the current one. This extension's main task is to give information well-defined meaning, so that cooperation between computers and humans is enabled. By giving information meaning, the

Semantic Web also enables users to perform complex queries as well as automatically finding, combining and acting upon information in a way that previously required human input and thought [5]. For example, let us say that we want to buy some books online with the criteria that they are all new and hardback editions. Additionally, we want the books at the cheapest available prices at a location close to us so that shipping is fast.

Without the Semantic Web we would have to do all the work ourselves. That is, browsing various retailers, comparing item prices, shipping rates and times. However, with the Semantic Web we could input all our criteria into a computerized agent as a query. The agent would then search the Web and find the best results which would then be returned to us. Furthermore, the agent could learn from previous experience, so that if we have a particular good or bad experience from a site, the agent would take note of this for future queries. This way, the agent could exclude the bad site from the result set, or rank the good site higher if that site is applicable to the query [5].

As we see, the Semantic Web could be an excellent tool for helping with the information overload problem because in a fully realized Semantic Web we could easily find exactly what we are looking for and not have to look through a potentially huge number of results that a normal search engine produces. However, the Semantic Web is still just an idea under development. Critics claim that the Semantic Web is a Utopian idea and that a true semantic web is an AI-complete problem that has no technological solution [88][31].

XML and RDF

Naturally, the Semantic Web needs a set of technologies to work. Two of those are the eXtensible Markup Language (XML) and the Resource Description Framework (RDF) [5]. XML lets everyone make their own tags that annotate web pages or sections on a page. These tags are hidden from the end user that is viewing the page and allows for arbitrary structuring of documents. However, the XML tags say nothing about what the structures mean. For that we need RDF.

RDF expresses information meaning encoded in sets of triples. These triples consist of a subject (a person, a web page), a property or predicate ("is the father of", "is the creator of") and lastly an object (another person, another web page, etc). All of the elements in a triple are identified by a Universal Resource Identifier (URI). The most common type of URI is the URL which points to a web page, but an URI can also point to objects that are not on the Web, such as books or other appliances. Since the properties are also identified by URIs, anyone can define a new concept by defining a URI for it somewhere on the Web [5]. Since RDF uses URIs to encode information about related things in a document, the URIs ensure that the concepts are tied to a unique definition that everyone can find on the Web.

Ontologies

However, this is not the end of the story. It is possible for a concept to have several different identifiers. Therefore, the Semantic Web must also include ontologies, so that common meanings can be discovered. An ontology in the context of the Semantic Web is a document or file that formally defines the relationship between terms. The most typical kind of ontology has a taxonomy and a set of inference rules [5].

The taxonomy defines, as mentioned, classes of object and the relationships among them. In such a taxonomy we can define that an *address* is of type *location*, and that *city codes* can only apply to *locations*, and so forth. It is possible express a large number of relationships among entities by assigning properties to classes and allowing subclasses to inherit these properties. If we assume that the concept *city codes* must be of type city and that every city has a web site,

then it is possible to discuss a web site associated with a *city code*, even if the two are not directly linked to each other through, for example, a database [5].

Inference rules make the ontologies even more powerful. We can for example have the rule: *"If a city code is associated with a state code, and an address uses that city code, then that address has the associated state code."*. A computer can then derive a Massachusets Institute of Technology address, being in Cambridge, must be in the state of Massachusets, which is in the United States of America, should be formatted to US standards. While the computer does not "understand" this information in the way we humans do, it can still manipulate it in a way that is meaningful and useful to us [5].

The Semantic Web Stack

Even if it has been under development for many years, a standardization of the Semantic Web is still under development. As we can see in figure Figure 2.4 there are many components that must be in place for the Semantic Web to work. In addition, some of these components and layers are not yet fully realized.

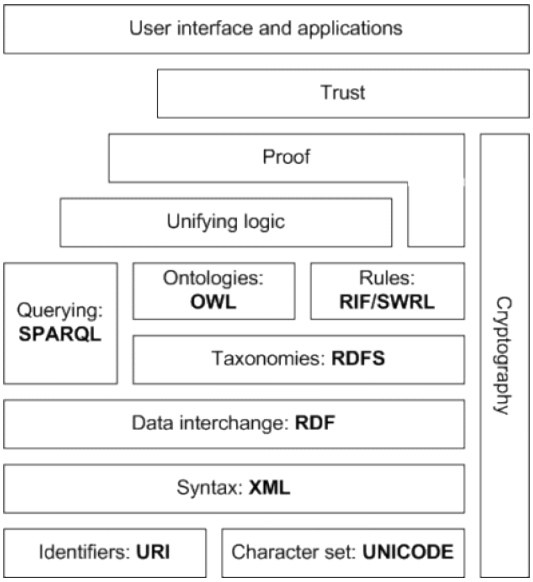

Figure 2.4: The Semantic Web stack

2.3.3 Debategraph

Debategraph is a web-based concept mapping and mind mapping software that focuses on learning about, thinking through and deciding upon complex public issues [23]. The software provides visualization tools to help communities think through topics by building and sharing dynamic, editable and rateable maps of subjects from multiple viewpoints. It aims to increase the transparency and rigor of global political debate by making the best ideas and arguments on all sides of a public issue freely available to anyone [23]. The creators of Debategraph claim that public debate often suffers from confusion and high noise-to-signal ratio due to rhetoric, obfuscation, digressions and repetitive contributions. Their goal is to address these problems by providing to possibility to create comprehensive and succinct maps of complex debates where instances of

repetitive clutter and ”noise“ are non-existent [23]. Figure 2.5 shows an example of what a part of a debategraph can look like.

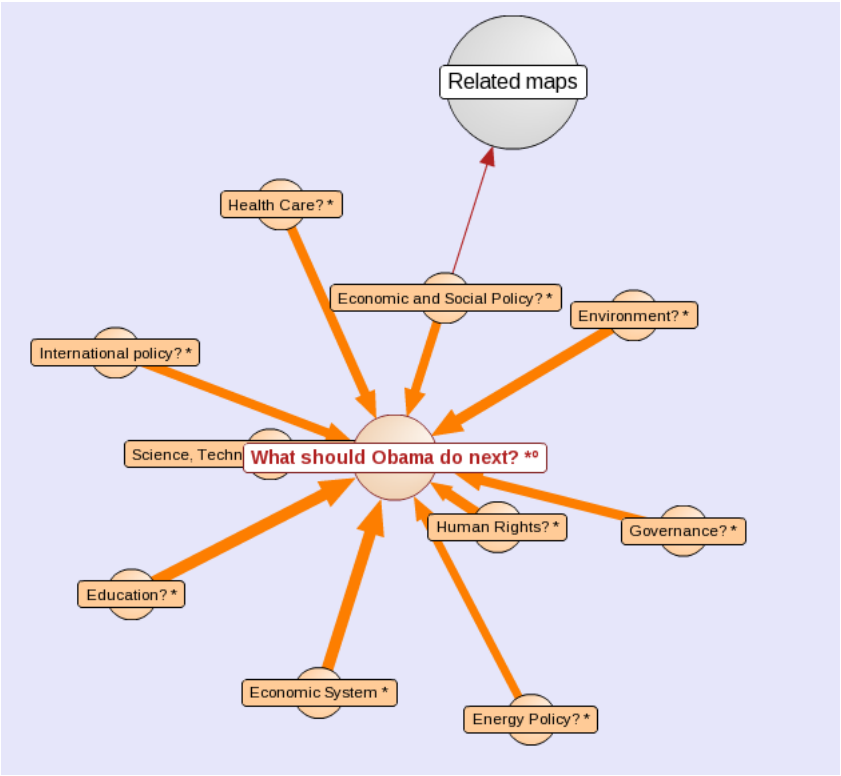

Figure 2.5: A debategraph example

In Debategraph, building a map consists of three steps. First, a subject needs to be broken down into meaningful ideas, then the relationship between the ideas must be figured out. Lastly, the ideas and the relationship must be expressed visually. Figure 2.6 shows the basic elements, or building blocks, of a map in Debategraph. The first element, the largest circle, is the Issue. Issues are usually questions that are raised about a subject and can be for example *“What should Obama do next?”*. An Issue can then have either sub-issues or Positions tied to it. A Position is a suggested response to an Issue. Then, each Position can have a number of Supportive or Opposing Arguments attached to it. Each Supportive or Opposing Argument may also have Arguments attached. Each building block has an associated color in order to differentiate them at a glance [23].

2.3.4 Knowledge Cartography

Knowledge Cartography was invented by Marco Quaggiotto and is an ongoing research that aims to give a cartographic approach to the representation of knowledge [85]. The goal of the research is to extend the cartographic metaphor beyond visual analogy and create a tool that allows for representation of knowledge that can be likened to geographical maps. Just like geography, the world of knowledge is complex, heterogeneous and dynamic. The map that is created in Knowledge Cartography is not supposed to be a passive representation of reality, but a tool for the production of meaning. In other words, the map can be regarded as a communication

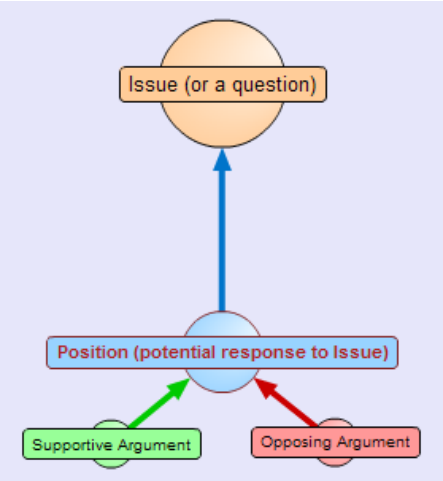

Figure 2.6: The elements of Debategraph

device that has its own language, rhetoric and tools [85]. A Knowledge Cartography map can for example be used to go into sub-areas of the knowledge representation space in order to gain a better understanding of issues, or find connecting paths between seemingly separated research areas. The map can also be used with malice to hide, conceal or falsify reality [85].

Currently, a piece of software based on the Knowledge Cartography concept is under development. This software is currently called ATLAS. The name refers to the "atlas of knowledge spaces", and is a system of representations of space and research in complex contexts rather than a list of maps, which a geographical atlas usually is. ATLAS is being developed for the management of research systems and is designed to support common research tasks such as surveys, mapping and analysis [84]. The main elements in ATLAS are resources, actors and relationships. ATLAS uses these elements to visualize how they interact in the creation of new knowledge [85][84].

The ATLAS software allow users to build his or her own bio-bibliographic database using four types of resources related to their research; Authors, texts, projects and research groups. Each resource in the system can be described in two ways. Either collectively by users in its fundamental features, such as description and location, or by each individual user through comments, tags or relationships between entities, such as relating a text to an author or an author to a project. Using this management model, it is possible to create a network of users, resources and keywords [84]. Further, by using the data provided by the system's users, maps of various levels of scale can be created. For example, one could create an entirely personal research map or a research community could create a collective map that every member has access to.

Obviously, ATLAS will also have a problem with information overload and density if one tries to add all entities in a rather large database to a map. In all likelihood, the resulting map will be entirely useless. ATLAS' solution to this is to use the cartographic rhetoric which says that *"reality must be selected, projected and symbolized in order to create maps with specific objectives"* [84]. Thus, ATLAS' users must themselves pick the data that creates that map that meets the intended need. The creation of a map consists of three steps. First, the user must select the relevant elements to be shown according to the maps intended use. For example, a user can choose to show the relationships between a set of authors and their texts. Then, the user must place the cartographic entities on the map space in a way that creates the best possible view that meets the intended function. This is called the projection process, and ATLAS provides four

main projection modes; Semantic, socio-relational, geographic and temporal spaces. Lastly, we have the symbolization process which allows the users to create a hierarchy of visual resources and highlight a subset of those resources if needed [84].

As we can see, the power of ATLAS lies in that it is possible for the users to intervene in the cartographic process as well as the navigational space, which is not possible in traditional maps.

Figure 2.7 and figure 2.8 show two examples of map types that are possible to create with the ATLAS software.

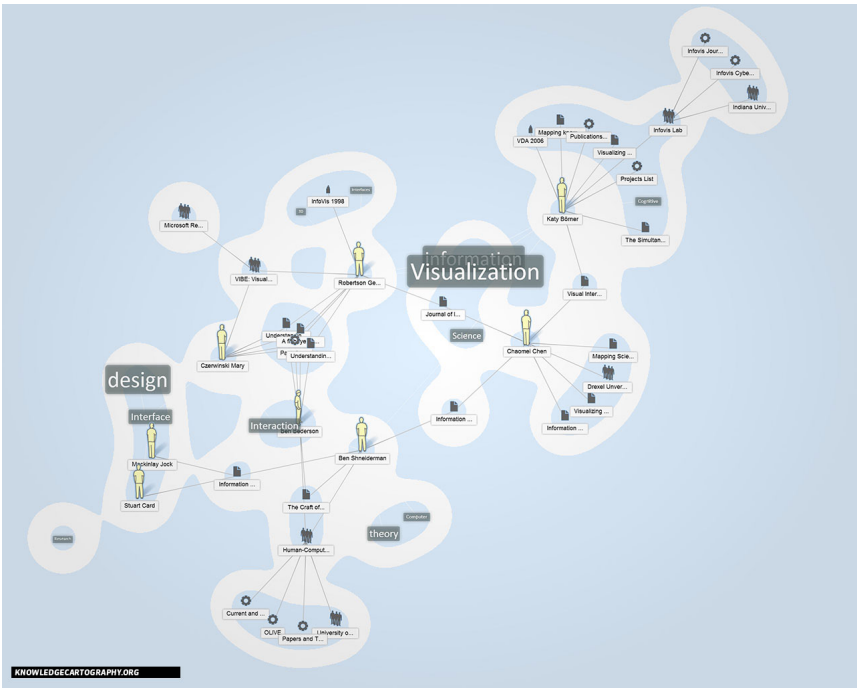

Figure 2.7: An example of a relational resource map in ATLAS

2.3.5 SpicyNodes

SpicyNodes is a Web-based system that allows its users to create radial maps that display hierarchical data [27]. A SpicyNodes map consists of a central node that is orbited by a number of related child nodes. Each child node can again be linked to other child nodes. When using a map, the user navigates through the nodes by clicking on a child node which then becomes a focus node. The focus node displays rich information and is surrounded by nodes that display related information [27]. Needless to say, these nodes are the children of the focus node. As the user navigates the map, a root path appears that traces back to the original central node. In order to avoid cluttering the screen with nodes, only parent and child nodes are displayed. That is to say, the focus node, its parent, so that the path back can be traced, and its children, in order to further browse the map, are the only nodes that can be viewed at any single time. Obviously, this causes nodes to appear and disappear while the layout rearranges itself as a user browser the map [27]. This kind of visual representation of information is intuitive and correspond to real-life phenomena and are useful to people who learn by *seeing* and *doing* as opposed to reading [27].

The creators of SpicyNodes conducted experiments to see if this claim was true. In fact,

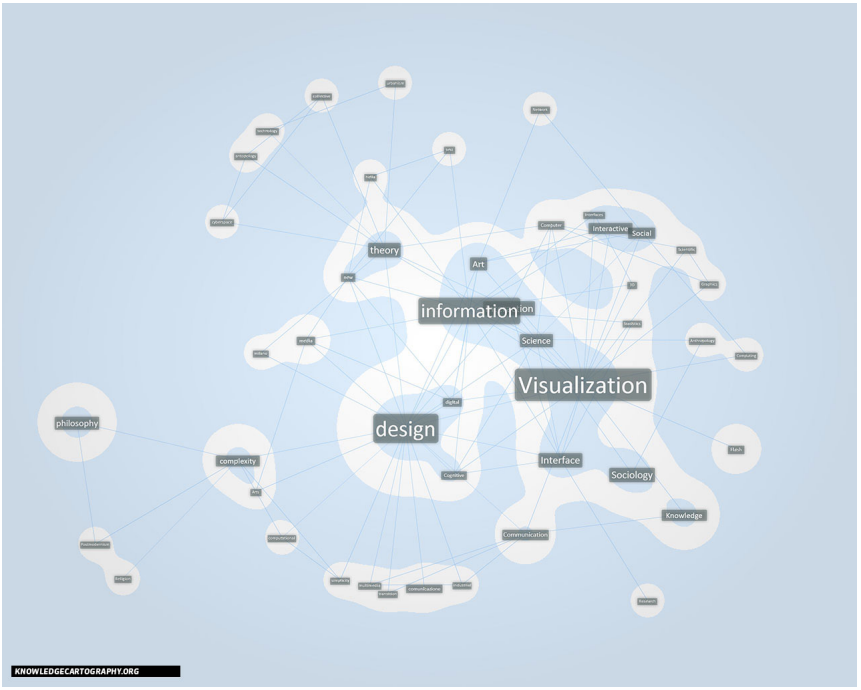

Figure 2.8: An example of a semantic map in ATLAS

they found that using radial maps is particularly effective for those who learn by performing a physical activity and remember visually [27][26]. In addition, SpicyNodes maps' radial layout mean they are effective for browsing to find specific information. Since the number of nodes increases exponentially with the number of orbits and we navigate a space with X^N nodes, we can find the information we are looking for in N navigational clicks, where X is the average number of nodes per orbit [26].

SpicyNodes has many different uses. It can be used as a personal brainstorming tool, as a site map for a web site, or as a tool that allows people to find what they seek in a domain. Figure 2.9 shows an example of a SpicyNodes map.

2.4 An Introduction to Reputation Systems

This section will introduce reputation systems, what they are and what they are normally used for. Concepts from reputation systems are integral to the affordances the Value Matrix Object provides. Specifically, reputation systems allow us to compute the quality and value of all knowledge work contributions, as well as the value of knowledge resources.

2.4.1 Characteristics of Reputation Systems

The purpose of reputation systems is to collect, compute and publish reputation scores for a set of objects or entities in a domain. These objects can be anything, such as users, goods or services. A reputation system's users are able to rate objects on a scale which can be binomial or multinomial. Ratings are collected by the reputation system and then used by a specific

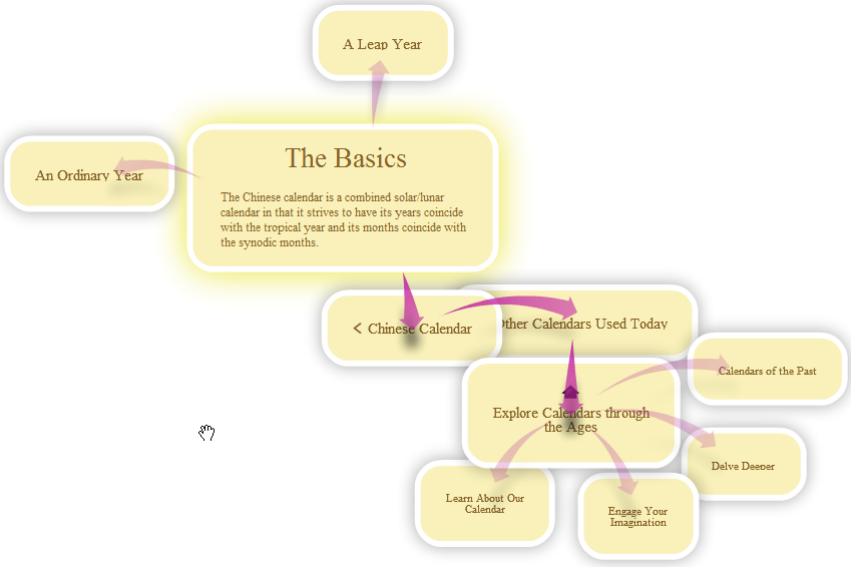

Figure 2.9: An example of a SpicyNodes map

algorithm that computes a score for the rated object. This score is published and made available for the system's users to see [90].

The basic idea behind these scores is that users can use them to make a decision whether to trust an object or not. This is useful in for example an Internet auction setting, where a person A can look at person B's reputation score before making a transaction with B [90]. It is in the interest of an object to have a high reputation score because they will most likely draw more attention and business than objects with a low reputation score. If an object has a low score, it means that the community perceives the object as low quality. Alternatively, an object with a high score is regarded as having high quality [90]. Naturally, an object's reputation score is given by a function of all incoming ratings, and can therefore quickly change. This means that if an object with a high score is suddenly given a string of negative ratings, its score can quickly start to deteriorate. The reverse is also true, an object that has a low score can have its score increased if it becomes praised.

Often it is hard to make the distinction between the concept of reputation and the concept of trust, but there is an important difference between the two. Trust is a subjective decision where you can choose to trust someone *despite* their bad reputation or the opposite: Trust someone *because* of their good reputation [51]. Mahoney defines trust as "a measure of willingness to proceed with an action (decision) which places parties (entities) at risk of harm and is based on an assessment of the risks, rewards and reputations associated with all the parties involved in a given situation."[71]. On the other hand, Mahoney says that reputation is a measure of the trustworthiness of an object based on the collected and processed ratings from a reputation system's users.

Resnick et al. claims that for a reputation system to work there are three properties that it must have. Firstly, entities must be long lived. This means that an entity cannot easily change its identity in order to erase past behaviour. Secondly, ratings about current interactions must be captured and distributed. In other words, ratings cannot be lost or ignored and there must be an incentive for system users to provide ratings. Lastly, ratings about past interactions must

guide decisions about current interactions. If the system's users do not care about the ratings, then the system has failed [90].

2.4.2 Reputation System Architectures

The general architecture of a reputation system is shown in figure 2.10. In the figure we have two actors, the *rater* and the *ratee*. After a transaction, the rater submits a rating for the ratee. The ratings are gathered by the *collector* and distributed to the *processor*. The processor uses an algorithm to compute the ratee's new reputation score based on the rater's newly submitted rating and previous ratings collected from other uses. Lastly, the *emitter* makes the reputation score publicly available [93].

This architecture can be split further down into two main types: The centralized architecture and the distributed architecture. The difference between the two is how ratings and reputation scores are distributed between the actors in the system [51][93].

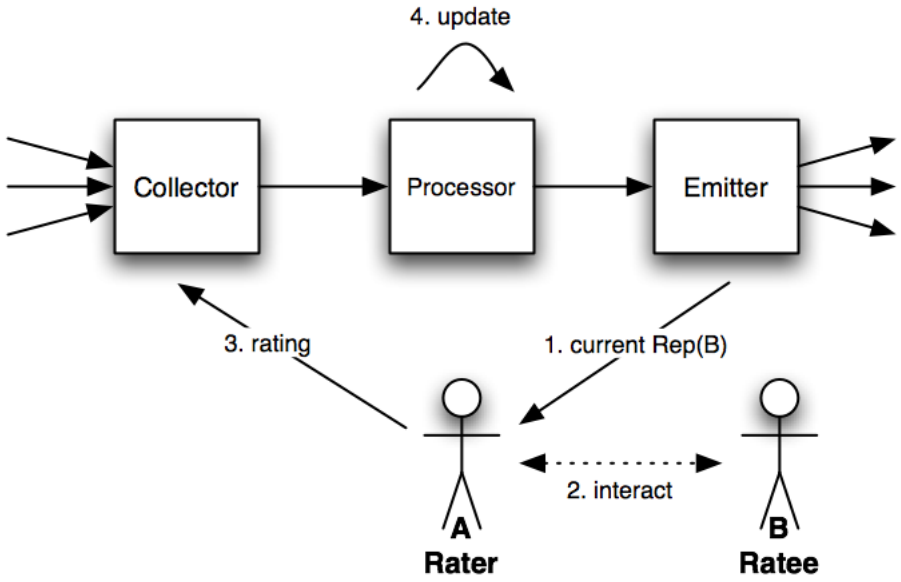

Figure 2.10: General reputation system architecture

Centralized Reputation Systems

In a centralized reputation system there is a central authority, typically called *reputation centre*, which is responsible for the jobs of the collector, the processor and the emitter seen in figure 2.10. The collector and the emitter are a part of the system's centralized communication protocols which allow users to provide their ratings to the reputation system. The protocols also make it possible for users to retrieve the reputation scores of the reputation system's objects [51]. In other words, all information must go through the reputation centre. A centralized reputation system also has a reputation computation engine which the processor is a part of. Its responsibility is to continuously compute reputation scores for the system's objects whenever new ratings are collected [51].

Figure 2.11 shows a typical centralised reputation system. A shows a history of past transactions, while B shows a possible present transaction.

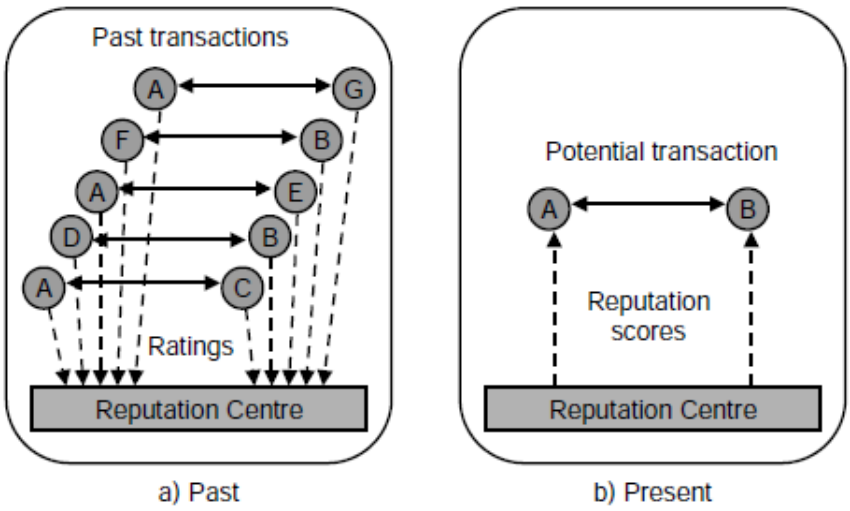

Figure 2.11: How a centralized reputation system works

Distributed Reputation Systems

A distributed reputation system differs from a centralized reputation system in that there are no central authority that collects ratings and provides reputation scores. Instead, there are either distributed hubs where system users can submit their ratings and request reputation scores, or the users themselves store their own ratings about the system's objects and provide them upon request from other users. Either way, the system's users must themselves find the ratings for the objects they wish to transact with and compute the reputation score for these [51]. Figure 2.12 shows a typical distributed reputation system.

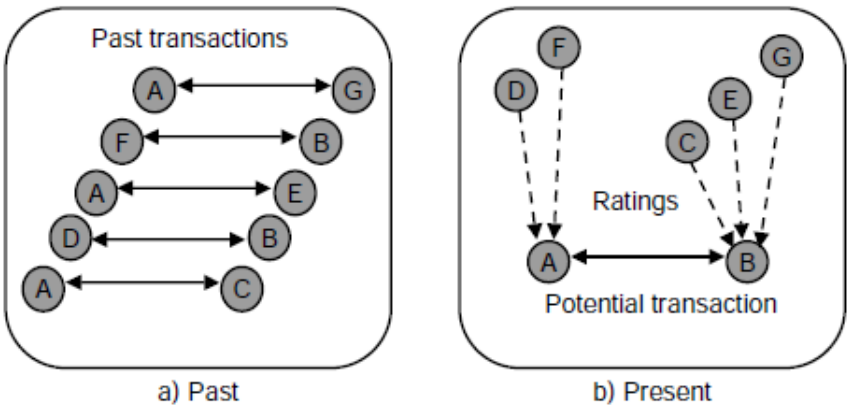

Figure 2.12: How a distributed reputation system works

It is perhaps not clear in what environments a distributed reputation system is better suited than a centralized reputation system. Peer-2-peer networks, such as torrent networks, is an

example of an environment where a distributed reputation system would be suitable.

2.4.3 Reputation Score Algorithms

We have so far talked about reputation systems, but not how they calculate the reputation score of an object or entity. Obviously, the reputation score of an object should reflect the community's general opinion about it, but private information can also be taken into account. Private information can for example be first hand experience or knowledge about an object and is often regarded as more reliable than third party opinions when computing the reputation score [51]. In this section, we will discuss various ways of computing reputation scores.

Summation of Ratings

The perhaps simplest way of computing an object's reputation score is to subtract the sum of negative ratings from the sum of positive ratings. While the advantage is that this system is easy to understand, it has a number of shortcomings. For example, let us say that a user has 150 positive ratings and 20 negative ratings. This would mean that the user's reputation score would be $150 - 20 = 130$. Then, there is another user who has 130 positive ratings and 0 negative ratings. Intuitively, the latter user would appear to be more reputable than the former user, but the summation of ratings algorithm makes no distinction between the two [75].

Although it makes sense to have a binomial scale with the summation of ratings algorithm, it works just as well with a multinomial scale. The only difference is that with a multinomial scale there are several degrees of negative ratings and positive ratings. For example, the multinomial scale could go from -5 to 5 instead of the -1 and 1 in a binomial scale. However, having a multinomial scale might be redundant. According to Resnick et al. empirical studies show that less than 1% of all ratings provided in a summation of ratings reputation system are negative and about 99% are positive [89][91]. Most likely, users will provide ratings that are on the extreme ends of a scale and the use of the other values will be minimal.

The studies also highlight another problem with the summation of ratings algorithm. It was found there is a high correlation between buyer and seller ratings. In other words, if participant A gives participant B a positive rating, there is a high chance that participant B will give participant A a positive rating as well. The same was found to be true for negative ratings [91].

Normally, when computing the reputation score of an object, each rating is pure. This means that it has not been changed by the system since it was submitted by a user. However, it is possible to give each rating a weight. This weight could be determined by for example the rater's reputation, how long it has been since the rating was submitted, the difference between the rating and the current reputation score and others. Using such weights could enhance the robustness of the reputation system [51].

Average of Ratings

There has been proposed slightly more advanced algorithms, such as the average of ratings algorithm by Schneider et al. [115]. The idea is to summarize all the positive ratings and divide them by the total number of ratings in order to get the reputation score. If we use our example from above, the first user will have a reputation score of $150/170 = 0.88$, while the second user will have a reputation score of $130/130 = 1$. We remember from the summation of ratings algorithm that the two users appeared equally reputable. However, when using the average of ratings algorithm, the second user is given a higher score and will rightly appear as the most reputable user of the two.

Example of Results Using Wilson Score Interval		
# Positive Ratings	# Negative Ratings	Score
250	100	0.6648317184611
1000	500	0.6424116916199
100	50	0.58789756740385
1	0	0.20654931654388

Table 2.1: Example of results using the Wilson Score Interval

The average of ratings algorithm works fine if all objects in the system always have a high number of ratings. Its weakness lies in the scenarios where an object has few, but positive ratings and another object has many positive ratings and a few negative ratings. For example, an object A has 2 positive ratings and 0 negative ratings and object B has 200 positive ratings and 2 negative ratings. The average of ratings algorithm will put object A above object B when sorting on reputation, which clearly should not be the case [75].

Similarly to the previous algorithm, this algorithm can also make use of giving individual ratings a weight.

Wilson Score Interval

A better way to compute the reputation score of an object is to use the lower bound of Wilson score confidence interval. The problem that the two above algorithms have is that they do not balance the proportion of positive ratings with the uncertainty of a small number of observations, i.e. total ratings. We can use the formula created by Edwin B. Wilson to solve our problem [110].

$$\frac{\hat{p} + \frac{1}{2n} z^2_{1-\alpha/2} - z_{1-\alpha/2} \sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2_{1-\alpha/2}}{4n^2}}}{1 + \frac{1}{n} z^2_{1-\alpha/2}}$$

Here $\hat{p}$ is the observed fraction of positive ratings, while $z^2_{1-\alpha/2}$ is the quantile of the standard normal distribution, and lastly, n is the total number of ratings [110][75].

A system that uses the Wilson Score Interval method for computing reputation will use the above formula on all objects, and use the result to sort the objects from highest to lowest score. Table 2.1 shows an example where four samples have been taken from a binomial rating system. We assume that we have some statistics that give $z^2_{1-\alpha/2}$ and that the confidence level of our lower bound is 97.5%. We see that the item with 250 positive and 100 negative ratings is ranked at the top, while the item with 1 positive rating and 0 negative ratings is at the very bottom of the list. If we had used an average of ratings method, the latter item would be ranked at the top.

Obviously, Wilson's solution only works for binomial ratings, i.e. either entirely positive or entirely negative ratings. Solutions for multinomial proportion confidence interval exists, and R.G. Miller has conducted a survey over many such methods [77].

Bayesian Systems

Bayesian computation engines share some similarities with the Wilson Score Interval as both uses probability to predict the outcome of future ratings. Both binomial and multinomial ratings are supported using a Bayesian system and we will here discuss both versions.

Binomial Bayesian Reputation Binomial Bayesian reputation systems use the beta probability density function as an underlying statistical foundation when computing reputation scores for the system's entities. The beta PDF can be expressed using the gamma function Γ as [51]:

$$Beta(p|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}p^{\alpha-1}(1-p)^{\beta-1}$$

where $0 \leq p \leq 1$ and that α, β, which are the two parameters in the binary state space, must be greater than 0. There is a restriction however, which says that the probability variable $p \neq 0$ if $\alpha < 1$ and $p \neq 1$ if $\beta < 1$. Lastly, the gamma function is an extension of the factorial function and looks like the following:

$$\Gamma(n) = (n-1)!$$

The probability expectation value of the Beta distribution is what is used to represent the reputation score of an entity and is given by [51]:

$$E(p) = \frac{\alpha}{\alpha+\beta}$$

Reputation is computed by statistical updating of the beta PDF. That is, the updated, a posteriori, reputation is continuously computed by combining the previous, a priori, reputation with every new rating. At the beginning of an entity's lifetime it has not received any ratings. Thus, the a priori distribution is the beta PDF with $\alpha = Wa$ and $\beta = W(1-a)$. Here, W is the non-informative prior weight and a is the base rate of the positive outcome in the binary state space. The normal base rates are $a = 0.5$ and $W = 2$. When an entity has received r positive ratings and s negative ratings, we compute the a posteriori distribution using the beta PDF with $\alpha = r + Wa$ and $\beta = s + W(1-a)$. By inserting these values into the probability expectation value function we can compute the reputation score S. The score S is the probability that the next rating the entity will get is positive [52]. S is given by:

$$S = \frac{(r+Wa)}{(r+s+W)}$$

Multinomial Bayesian Reputation In a multinomial Bayesian reputation system there are k different levels over which ratings can be provided, as opposed to a binary state space [52]. If we let these k levels be a set $\Lambda = L_1, ..., L_k$ and assume that ratings are votes on the elements of Λ, then we can use the Dirichlet probability density function over the k-component random probability variable $\vec{p}(L_i), i = 1...k$ with sample space $[0,1]^k$, subject to the simple additivity requirement $\sum_{i=1}^{k} \vec{p}(L_i) = 1$ [52].

The Dirichlet distribution captures a sequence of ratings of the k possible outcomes with k positive real rating parameters $\vec{r}(L_i), i = 1...k$. To use the Dirichlet function we define two vectors. $\vec{p} = \{\vec{p}(L_i)|1 \leq i \leq k\}$ which denotes the k-component probability variable, and $\vec{r} = \{r_i|1 \leq i \leq k\}$ which denotes the k-component rating variable. In addition, we have a base rate vector $\vec{a}$ over the state space that represents prior, the a priori, information. The Dirichlet function is now given by [52]:

$$Dirichlet(\vec{p}|\vec{r},\vec{a}) = \frac{\Gamma(\sum_{i=1}^{k}(\vec{r}(L_i)+W\vec{a}(L_i)))}{\Pi_{i=1}^{k}\Gamma(\vec{r}(L_i)+W\vec{a}(L_i))}\Pi_{i=1}^{k}\vec{p}(L_i)^{\vec{r}(L_i)+W\vec{a}(L_i)-1}$$

$$where\begin{cases}\sum_{i=1}^{k}\vec{p}(L_i)=1 \\ \\ \vec{p}(L_i)\geq 0,\forall i\end{cases} and\begin{cases}\sum_{i=1}^{k}\vec{a}(L_i)=1 \\ \\ \vec{a}(L_i)\geq 0,\forall i\end{cases}$$

We can now express the multinomial reputation score vector $\vec{S}$, which contains the probability expectation values of the k-random probablity variables, as [52]:

$$\vec{S}(L_i) = E(\vec{p}(L_i)|\vec{r}, \vec{a}) = \frac{\vec{r}(L_i) + W\vec{a}(L_i)}{W + \sum_{i=1}^{k} \vec{r}(L_i)}$$

We remember that W is the non-informative prior weight and is set to W = 2.

Discrete Trust Models

The idea of using discrete trust models in reputation systems spawned out of the perception that humans are better at rating performance using discrete verbal statements than using other measures, such as integers [51]. Various models have been proposed, but we will here use Abdul-Rahman and Hailes' model as an example [1]. In their model, a user's trustworthiness can be described as Very Trustworthy, Trustworthy, Untrustworthy and Very Untrustworthy. The model uses look-up tables, which has entries for referred trust from the referring agents and whether the referrals have been up- or downgraded. The entries are used to determine the derived trust of a user. After a transaction has occurred between two parties, the relying party's experience can be used to determine the trustworthiness of his transaction partner's referrers. The model assumes that personal experience reflects the real trustworthiness of a service provider, so that referrals about the service provider that differ from the personal experience suggests that the referring agent underrates or overrates. The referrals from those agents who underrate will be upgraded, while the referrals from those agents who overrate will be downgraded [1][51].

Flow Models

A flow model is a system that computes reputation score by transitive iteration through looped or long chains [51]. There are two variants of the flow model. The first assumes that there is a constant reputation weight for the whole community that is distributed among the users in the community. A user can only increase his or her reputation at the expense of others. The reputation of a user increases as a function of incoming flow and alternately decreases as a function of outgoing flow [51]. Systems that use this kind of flow model include Google's PageRank, which is described in Section 2.4.4.

The other flow model variant does not require the sum of reputation scores to be constant, but rather looks for reputation score for all users to converge to stable values [51].

2.4.4 Examples of Real Systems

In this section, we will discuss a few examples of how reputation systems can be used on the Internet. We do this because we can learn from them and use the knowledge in the Value Matrix Object design.

Reputation in Search - PageRank

PageRank, invented by Page et al, is an algorithm which purpose is to rank search results on their reputation [82]. The basic idea is that a page's reputation score is calculated from the number of hyperlinks pointing to that page. In other words, one such hyperlink can be regarded as a positive rating. Page et al. has defined the PageRank algorithm as follows [82][51]:

> Let P be a set of hyperlinked web pages and let u and v denote web pages in P. Let $N^-(u)$ denote the set of web pages pointing to u and let $N^+(v)$ denote the set of

web pages that v points to. Let E be some vector over P corresponding to a source of rank. Then, the PageRank of a web page u is:

$$R(u) = cE(u) + c \sum_{v \in N^-(u)} \frac{R(v)}{|N^+(v)|}$$

where c is chosen such that $\sum_{u \in P} R(u) = 1$.

We see that the equation consists of two terms. The first term $cE(u)$ gives rank value based on initial rank, while the second term $c \sum_{v \in N^-(u)} \frac{R(v)}{|N^+(v)|}$ gives rank value as a function of incoming links to u. Page et al recommends that E is chosen such that $\sum_{u \in P} E(u) = 0.15$ [82][51].

Google is currently using the PageRank algorithm in their search engine, but the live algorithm is more complex than the one showed above. Its details are not publicly known, but it has been revealed that the live algorithm also take other elements into account when computing the PageRank of a page to make it difficult to manipulate on purpose [51]. Additionally, Google has said that PageRank is not the only metric involved when ranking a web page [97].

Reputation in eCommerce - eBay

eBay is a very popular Internet auction site. Once a transaction between two parties is complete, eBay offers the opportunity for the two parties to rate each other. The ratings can either be positive (1), negative (-1) or neutral (0). All ratings are collected by a central hub which also computes the reputation scores for every user. In other words, eBay uses a central reputation system [51].

To compute reputation score, eBay uses a summation of ratings algorithm. The sum of negative ratings is subtracted from the sum of positive ratings to get the total score. In addition, eBay provides the opportunity to view a user's behavior for three time windows: last six months, last month and last seven days [51]. This allows users to make better decisions about possible transactions since a user with a high number of received ratings and a good reputation score can recently have received a string of negative ratings or vice versa.

As discussed earlier, reputation systems that use a summation of ratings algorithm can be misleading because of the way scores are computed. In addition, it has been found that almost 99% of all ratings are positive [89]. However, eBay's reputation system seems to work because the users, both buyers and sellers, are honest in general. Obviously, there are scammers and abusers, but less than there might be in a marketplace that is open and unregulated [51][89].

Reputation in Online Communities - Slashdot

Slashdot is a message board where users can submit news articles and give comments on posted articles. The articles that are posted on the Slashdot site are chosen by the Slashdot staff, but the chosen are based on the submitted articles from the community. Obviously, this is done to avoid spam and low quality articles. Once an article has been made public by the Slashdot staff, both anonymous users and logged in users can read and comment on it [51]. Since anyone can write comments, they need to be moderated.

The Slashdot moderation system consists of two layers and therefore two types of moderators [51]. Type A moderators are allowed to moderate comments on articles. All users that are logged in can become type A moderators. These are automatically selected by Slashdot. Once selected, each type A moderator is given several moderation points and has three days to spend them. A moderation point is spent by rating an article comment. The rating is selected from a list of negative (off-topic, flamebait, troll, redundant, overrated) or positive (insightful, interesting,

informative, funny, underrated) adjectives [51]. Giving a positive rating to a comment will increase its score by 1, while a negative rating will decrease its score by 1. A comment's score is an integer value in the interval -1 to 5. Usually, a comment will start out with a score of 1, but this can vary based on the *Karma* of the user that posted the comment [51].

The purpose of the comment score is to allow users to choose what he or she is able to see when browsing Slashdot. A user may set a threshold so that he or she can filter out bad comments or spam by setting the threshold to 4 or 5. Setting the threshold to -1 will not filter out any comments, and thus everything will be shown [51].

There are six types of Karma values in Slashdot's system: Terrible, Bad, Neutral, Positive, Good and Excellent. Each registered user has a given Karma value which starts out at Neutral. It will either increase or decrease based on other user's moderation of your article comments. Having a high Karma value influences two things. Firstly, you will have more moderation points to spend when you are selected type A moderator, and secondly, your comments will have a higher initial score. If you have a low Karma value, your comments will have a lower initial score and you will have less points to spend when moderating others [51].

As mentioned, Slashdot has two types of moderators. The first one, type A has been described above. The other one, type B, has the role of moderating the type A moderators. Any user who have been registered on Slashdot for a given amount of time has the option to moderate the type A moderators' ratings any time he or she wishes. The type B moderator will decide if the type A moderator's rating was fair, unfair or neither. Moderation performed by a type B moderator will affect the Karma of the type A moderator [51]. The two moderation layers together tries to create a healthy moderation environment where good users are rewarded and bad users are punished. In addition, the Slashdot staff has the ability to use any amount of moderation points, so if Slashdot is affected by massive amount of spam and unfair ratings, the staff can try to stabilize the system by manually giving all the bad comments a negative rating.

The Slashdot system is not perfect, but it is still being tuned and modified to find the best way to reduce spam and bad quality content as well as promote good content so that Slashdot remains readable and useful [51].

Reputation in Programming Communities - Stack Overflow

Stack Overflow is a website that allows its users to post questions and answers on a wide range of topics in computer programming. The members of the website can vote questions and answers up or down. These votes are the integral part of the site's user reputation system. A user's reputation is a part of that user's identity on the site, and it determines the user's expertise level as well as the level of respect in the community the user has. Reputation is only gained whenever other users of Stack Overflow approve of the content a user provides [96]. In addition, a high level of reputation also gives those users more functionality that low-reputation users cannot access [96].

The quality of a user's interaction with the system and the community determines reputation gain or loss. As mentioned, the primary reason for reputation change on Stack Overflow is voting. A user who has his posts voted up receives and increase in reputation. Naturally, posts which are voted down impacts their authors reputation negatively. Upvotes are more heavily weighted than downvotes [96]. For example, a user receives 10 reputation points when one of his answers is voted up, but is only subtracted 2 reputation points when an answer is voted down. A user is also subtracted 1 reputation point whenever he downvotes an answer. There is currently a maximum of gaining 200 reputation points from upvotes per day [96].

Reputation in Product Review Sites - Amazon

Amazon is known to most people as an online book vendor that also sells other products. Their website also contains the functionality for site members to write product reviews. A product review consist of a prose text written by the reviewer as well as a rating in the range 1 to 5. This is where reputation comes in. Amazon uses an average of ratings scheme to compute a product's reputation score [51]. In addition, all site users, which includes non-members, may vote on reviews as being helpful or not helpful. The number of helpful votes as well as the total number of votes are shown together with each review. For example, a review might say "15 of 20 people found this review helpful". A product also displays how many total reviews it has as well as how many of each star it has been given. Lastly, users have the option of seeing the most helpful favorable review and the most helpful critical review [51].

The site ranks each reviewer based on the number of helpful votes he or she has received and posts a top 1000 list. While being a top reviewer does not carry any incentives, Amazon's review scheme has trouble with ballot stuffing where a reviewer's helpful votes have been artificially elevated or where a reviewer has been bad mouthed and thus fallen in ranking [51]. The problem is of course enhanced by that fact that any one, even those not logged in, can vote on a review. Amazon has tried to combat this by only allowing one vote per unique visitor per review. However, since unique visitors are logged via cookies it is easy to circumvent Amazon's solution by deleting the cookies or just using several computers [51].

Obviously, the products' reputation scores also suffers from the same problems that all reputation systems using an average of ratings scheme does, as discussed earlier.

2.4.5 Issues in Reputation Systems

Most, if not all, reputation systems has some kind of problem or flaw. We will here discuss the most common problems as well as presenting proposed solutions where applicable. This is important because these issues must be addressed by the Value Matrix Object as it employs a reputation system.

Incentives

One of the largest issues with reputation systems is that there are often no incentives to provide ratings [51]. Providing ratings does not reward the rater directly, but rather benefits the community as a whole. Using an Internet auction site as an example, a buyer has no personal incentive to give his or her transaction partner a rating, but it would help other buyers, i.e. the community, to decide whether or not to transact with that person in the future. On the other hand, sellers have an incentive to provide positive feedback to the buyers in hope that they will reciprocate the courtesy [17].

Although there might be low incentives for providing ratings, studies done on eBay's reputation system show that 60.7% of the buyers and 51.7% of the sellers provide ratings about each other [89]. However, in this particular case the sellers actually have an incentive to provide ratings to their transaction partners as mentioned. These numbers are likely lower in a system where users rate physical objects, such as books or similar, instead of rating other users. There has been proposed ways of addressing the incentive problem. For example, Miller, Resnick and Zeckhauser have proposed a mechanism where truthful feedback would be financially rewarded [76].

Positive Rating Bias

Another problem with reputation systems is that there is often a bias towards positive ratings. A study by Resnick and Zeckhauser in 2002 found that only 0.6% of the ratings buyers provided were negative [89]. The number of negative ratings from sellers were slightly higher. The same study claims that the reason for these low numbers can be explained by either general politeness causing people to provide positive ratings, or fear of retaliation if a negative rating is provided. Positive rating bias can possibly be counteracted by making the ratings anonymous [51].

Unfair Ratings

Unfair ratings, both unfairly positive and unfairly negative, is a fundamental issue and preventing their influence is very important for a reputation system to be credible [51]. There are two main methods of avoiding bias from both types of unfair ratings; Endogenous discounting and exogenous discounting.

Endogenous discounting of unfair ratings assumes that unfair ratings can be recognized by their statistical properties. The idea is to identify the assumed unfair ratings and then either exclude them entirely or give them a low weight when computing the reputation score of an entity [51]. There have been some proposals for handling this issue. Mao Chen has proposed a method where the system automatically computes reputation scores for users based on the ratings they and others give to entities [17]. These reputation scores are incorporated to generate value-added information about entities. The score of an entity is calculated as the average of all its ratings, but with the rater's reputation as a weight. In addition, Chen proposes to calculate also the confidence of a score, which is important in cases were only a few users with low reputation have rated an entity [17].

Dellarocas has also suggested a method for discounting unfair ratings [94]. First, he suggests that unfair ratings can partially be avoided by concealing the identity of the parties in a transaction. With the parties being anonymous, they cannot identify one another and has no reason to provide unfair negative ratings. However, Dellarocas also makes it clear that it is not possible to conceal both parties always, for example if one party is a hotel or restaurant [94]. Still, this anonymity scheme can be used in many other cases.

Further, Dellarocas argues that the parties in a transaction might be able to signal their identities to one another in order to perform ballot box stuffing or give unfair positive ratings. In other words, concealing the identities are only useful to prevent unfair negative ratings. However, it is possible to reduce the effects of unfair positive ratings. Dellarocas proposes the following algorithm [94]:

> "*To calculate an unbiased personalized reputation estimate $R_b(s)$, we first use collaborative filtering techniques to identify the nearest neighbor set $Nofb$. N includes buyers who have previously rated s and who are the nearest neighbors of b, based on their similarity with b on commonly rated sellers. Sometimes, this step will filter out all the unfair buyers. Suppose, however, that the colluders have taken collaborative filtering into account and have cleverly picked buyers whose tastes are similar to those of b in everything else except their ratings of s. In that case, the resulting set N will include some fair raters and some unfair raters. The ratings will, therefore, form two clusters: the lower cluster N_l, which consists of fair ratings and the upper cluster N_u, which consists of unfair ratings. To eliminate the unfair ratings we apply a divisive clustering algorithm, such as the one proposed by Macnaughton-Smith et al., in order to separate the set of raters N into two clusters N_l and N_u based on some function of their ratings of s. Finally, we calculate the final reputation estimate $R_b(s)$ as a function of the ratings provided by members of cluster N_l only.*"

Exogenous discounting of unfair ratings methods determine the reputation of a system's users externally to determine the weight given to each user's ratings [51]. These methods assume that users with low reputation are likely to give unfair ratings, while users with high reputation is more inclined to give honest ratings. Further, it is assumed that private information is more reliable than public information. That is, personal experience should be considered more reliable than ratings from a third party. If this is the case, then the reputation system that used exogenous discounting of unfair ratings can compare the private information to the public information in order to decide if the public information is reliable [51].

Some schemes that have been proposed include one from Buchegger & Le Boudec who's scheme classifies users as trustworthy or not trustworthy using a Bayesian reputation engine and a deviation test [8]. Additionally, Yu & Singh proposes a system that uses a specialized Weighted Majority Algorithm which gives a weight to each user [113][51].

Change of Identities

For a reputation system to be credible, it must make sure that identities cannot be easily changed in order to reset previous behavior. In other words, a user's received ratings should always be associated with the user even if the user changes his name. A reputation system would lose a lot of credibility if users could easily change their identities and start fresh whenever they experience a loss of reputation [51]. Such behavior is discouraged as it damages the community and corrupts the reputation system, making it unreliable. It is not easy to prevent such behavior without also punishing honest newcomers. A normal way of discouraging users from creating a new identity is to associate a cost with it. eBay has an indirect solution to this problem. First, a new user zero positive ratings, meaning that it is less likely that he or she will attract business over other users with many positive ratings. Also, eBay charge a fee for each transaction, so there is an economic penalty for unacceptable behavior [51].

Other systems might give newcomers a low weight for their ratings until they reach a certain reputation level. Alternatively, newcomers' ratings can be ignored altogether until the system deems them "trustworthy", and all their past ratings start to count. A system that uses the latter method is the Internet Movie Database, IMDB, for their top 250 movies list. Only votes from "regular voters" are counted for the list. However, the criteria for a user to become a "regular voter" has not been disclosed by IMDB [45].

The Time Factor

Introducing considerations to time is essential to create a dynamic reputation system. By regarding time as a factor when computing an entity's reputation score, positive change is enabled. In the real world, the quality of a service or a commodity can either increase or decrease as time goes by. By discounting past ratings, we can reflect the current quality of a service or commodity instead of its historical quality [51]. Let us say that we have a service whose quality was previously mediocre, but due to various factors has become excellent. If all ratings are still equally weighted, the service's true quality is not correctly represented. However, by discounting the past we get the desired effect.

Discounting of the past can be implemented in various ways. The methods include implementing a forgetting factor, an aging factor or a fading factor [50][9][8]. For example, it is possible to disregard a rating if it is sufficiently old. Alternatively, we can give all old ratings a low weight or give them progressively lower weight relative to how old the rating is. On the other hand, it is possible to introduce a longevity factor which gives a rating a time to live and is disregarded after that time has passed [49].

Obviously, discounting of the past does not necessarily only need to be determined from a function of time. It is also possible to use frequency of transactions or a combination of both time and transaction frequency [51][8].

Ballot Box Stuffing

Ballot box stuffing is not only a problem in political elections, but also on the Internet. The problem is common because few systems have any protection against it [51]. It allows for users to provide more than a legitimate amount of ratings. Usually, a reputation system only allows a user to rate an entity once in a lifetime or once per transaction, but exploiting the ballot box stuffing problem makes it possible for a user to manipulate an entity's reputation score by providing a large number of, often unfair, ratings [51].

There are several ways for a reputation system to protect against this exploit. One way is to associate a cost to each rating. Then, ballot stuffing becomes financially infeasible since users are only allowed to provide one rating per completed transaction. To avoid fake transactions, it is also possible to charge a fee for each transaction [51]. Another way is to allow only registered users to provide ratings and make the registration process hard to automate by using CAPTCHA's, IP logging or similar techniques, although these can be bypassed by sophisticated software. Alternatively, a system can provide each user only a limited amount of rating points that they can use like, for example, Slashdot does. Lastly, each users ratings can be given a weight based on their own personal reputation. However, this scheme is problematic because it punishes new users even if they are legitimate.

Chapter 3

Domain Map Object

> "Visualize this thing that you want, see it, feel it, believe in it. Make your mental blue print, and begin to build."
>
> – Robert Collier

In this chapter we will present the Domain Map Object, and a subset of the possible affordances it provides. Intuitively, we can associate the Domain Map Object with a filing cabinet, that is to say, a place for organizing and storing knowledge resources. The DMO can also be viewed as a map, or a collection of maps, whose purpose is to show a high-level overview of the subject domain so that what is worth seeing can be easily located. Thus, one of the purposes of the Domain Map Object is to help explorers not get lost in the jungle of information resources. We will first provide a general overview of the DMO, discussing its roles and purposes. Then, we will present a subset of affordances it provides and practices it supports. Finally, we will discuss the components a domain map consists of.

3.1 General Overview

As we discussed in Chapter 1, The Domain Map Object is a boundary object for systemic innovation. From the viewpoint of a community of system builders, the DMO encapsulates tools created by tool making communities and exports the kind of functions the system builders need to create new sociotechnical systems. For the tool makers, the DMO is a set of desired affordances and work patterns provided by the system builders that the tool makers can make concrete technology out of [59]. The idea is that cross-disciplinary collaboration is required to create systems that aim to solve information overload and other complex issues.

Additionally, The Domain Map Object provides affordances with the means to organize information, giving it context and meaning. Information is organized by people in a community of interest, who add knowledge resources to the map and associate them with already existing subjects. This action makes the resources available to other members of the community. Anyone in the community can provide their insights and opinions about the resources that are located within the domain map, thus creating a collaborative and social way of organizing knowledge. A domain map is not only for mapping and organizing existing knowledge. By showing areas where knowledge is lacking, it can also encourage new research. We consider the activities the Domain Map Object supports as good practice since it is a step away from the old practices that focuses solely on volume production, which breeds information overload. Thus, by providing

the suitable affordances, we can say that the Domain Map Object, as a tool, can be used as a building block in sociotechnical system design.

A key point in sociotechnical systems is to provide motivation for the people, which can be in the form of either intrinsic or extrinsic rewards [78][83]. The Domain Map Object facilitates such reward systems by providing the affordances needed to carry out knowledge work activities. If users are encouraged and have incentives to contribute to a domain map, a healthy ecology, where the best contributors and the best knowledge resources are rewarded and brought forward, will emerge over time. Note that the Value Matrix Object, which we will discuss in the next chapter, is also an integral part of developing reward systems where all knowledge work activities are suitably rewarded.

The rest of this chapter will focus on the affordances and work patterns the Domain Map Object provides, as well as the elements a domain map consists of. That is, we will present and describe its functions and elements.

3.2 Affordances and Practices

The following is a subset of affordances and functionality that the Domain Map Object exports and provides [59]:

- **Visual representation**. As mentioned, a Domain Map Object is a visual representation of a domain. Since this representation is abstract, it is able to not only represent the knowledge that already exists, but also the knowledge which may not yet exist that belongs to the domain. This kind of visual representation facilitates learning and research. Additionally, a domain map offers multiple views of its domain, for example present the domain in various levels of detail. A fitting analogy is of course that of geographical maps which can be physical, political, climatological, and so on, and can exist in a variety of scales [59][85]. While this functionality is not elaborated upon in this thesis, the theory will originate from Quaggiotto's Knowledge Cartography that we presented in chapter 2.

- **Putting resources on the map**. This is one of the key aspects of the Domain Map Object. In the knowledge-work ecology the Domain Map Object provides affordances for, publishing new knowledge resources without adding it to a domain map can be viewed as littering since it leads directly to information overload. Therefore, federating the knowledge resources is good practice and is supported by the DMO.

- **Putting questions on the map**. In addition to putting resource on a domain map, research questions can also be added. This encourages collaboration with researchers that does not necessarily know each other, as well as cross-disciplinary discussions and research.

- **Querying the map**. The Domain Map Object provides ways to query the map and its resources in order to find and access the knowledge the resources contain.

- **Subscribing to map areas**. With this functionality, users are able to subscribe to specific areas on the map that they are interested in. By doing this, the users will be notified when new resources are added to that area or a significant update has occurred.

- **Federating insights**. Only putting resources on a map in an organized manner is not enough to eliminate information overload. Some way of federating insights and opinions is also imperative, so that essential knowledge is brought forward. Naturally, as it is in all communities, members tend to disagree about many issues. This is fine because it enables us to have a balanced ecosystem through the federation of each member's insights.

- **Exporting insights**. Through the federation of insights, the community will eventually arrive at a set of essential, or core, knowledge resources. We imagine that the DMO can export this knowledge to other domains by making it available through an "outbox". This "outbox" should be only be accessible by other research communities, but also by the general public so that they access the essential knowledge that the various communities have generated.

- **Importing insights**. Much like the above "outbox", the Domain Map Object will also make available an "inbox" where other communities may place insights and knowledge that they think may be of interest to the community that owns that domain map. The Domain Map will, if found suitable by the community, automatically incorporate the suggested insights into the domain.

The above functionality enables a set of processes and practices that is supported by the Domain Map Object [59]:

- **Organizing knowledge resources in a community-specific way**. Since the domain map is owned by a community, it also represents the way the community sees the world. Communities are necessarily different, and thus their domain maps will look different, but this does not mean that they cannot learn from one another by providing essential, federated, insights to each other.

- **Organizing knowledge in a context-specific way**. If we understand the domain map with a filing cabinet, then each subject on the map is a drawer in which knowledge resources can be associated. By organizing knowledge, we make it visible and available on the domain map.

- **Life-long and flexible learning**. The map provides guidance and makes relevant resources available through queries. Since a domain map will expand with an active community, it will always provide new knowledge resources to its users.

- **Organizing and directing research**. By putting questions on the map, researches may be encouraged to explore these questions and thus expand the domain map with new knowledge.

3.3 Components of the Domain Map Object

There are two main elements are used to build a domain map. These elements are topics and resources. In addition, the Value Matric Object plays an important part, but works in conjunction with the Domain Map Object rather than being an element of it. The Value Matrix Object will be further discussed in the next chapter, but as we recall from chapter 1, it is an object that is assigned to each resource and its purpose is to gather all information related to the value of the resource it is attached to, throughout the resource's lifetime. The topics are the subjects to which resources are associated. Topics also contain the functionality to query the resources that are associated with them. Within a domain map, resources acts as containers for either knowledge resources, such as research articles or papers, or human resources. That is, the digital profile of a community member. Both types of resources have a value matrix attached to them.

Figure 3.1 shows an example of a domain map. In this view we see five topics and two buttons that can be pressed. By clicking on one of the topics, the resources that are associated with it will be presented. The two buttons have a quite different function. They are used to alternate

between two views; The knowledge resource view, which is the active view in figure 3.1, and the human resource view, which shows the profiles of the community members that are registered with the domain map.

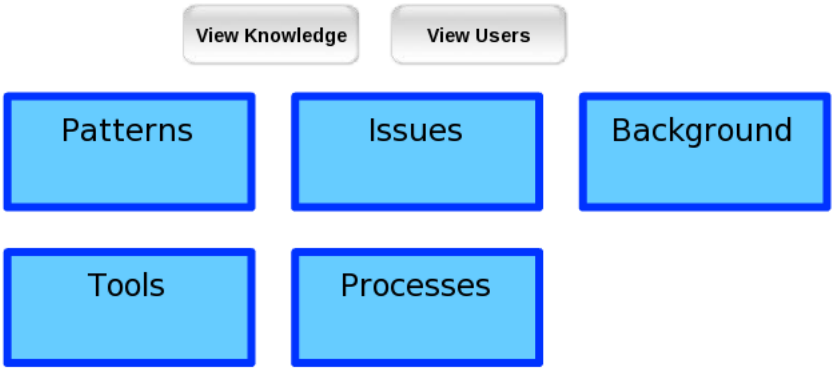

Figure 3.1: An example of a domain map

3.3.1 Topics

As we have established, the topics are one of the important elements of a domain map. The idea as that a set of topics are defined and added to the map by a selection of community members before the map is open up to the rest of the community. We do this in order to eliminate the risk of an overload of topics, something which could easily happen if all members could add topics at any point during the domain map's lifetime. However, nothing stops the community from adding new topics later, as long as this is agreed upon.

Alternatively, it is possible to open up the possibility for all members to create subtopics. This might be handy when a topic covers a very general area of interest and it is feasible to have more specialized subtopics. For example, the domain map might have a topic that covers reputation systems. Within that topic it is possible to have two subtopics "centralized systems" and "distributed systems" in order to distinguish between knowledge resources that contain information about either of those types of reputation systems. Naturally, a knowledge resource may be relevant in both of the subtopics, and in those cases the resource can be associated with both.

The topics in a domain map fulfill two purposes. The first is to be a "drawer" in our metaphorical filing cabinet. That is, a place put and store resources. Naturally, a resource may be associated with several topics if it is required. The other purpose is to provide context, i.e. a place on the map that can be visited, which is important not only for exploring the map, but also for querying resources.

3.3.2 Resources

The other elements of a domain map are the resources. We mentioned that a resource can either be a knowledge resource or a human resource. A knowledge resource can be anything that contains information that someone wants to index and file. The most obvious things are research articles and papers, but blog posts, journalist articles and interactive web tools among other things may also be resources. Because of this, a domain map cannot be viewed as a pure publishing tool, as it does not host the physical knowledge resource. Instead, the map just

provides an access point to it and focuses on the federation of the information and knowledge that is contained within it. To be clear, it is completely possible for an author to use a domain map as his only avenue for publishing his papers as long as they physically exist on some web server that the domain map can point to. However, since community members can add resources that they have not created themselves these resources must exist somewhere and the domain map must point to them, so the domain map does not act as a classical publishing tool. Instead, it encapsulates the knowledge resource and creates an access point to it. Furthermore, a value matrix is attached to the knowledge resource so that all activities related to its value can be logged. Lastly, the knowledge resource is associated, "filed", with one or more topics so that it can be found by browsing the map. As long as the knowledge resource exists on the map, its value matrix continuuosly gathers data.

A human resource is quite different. It is created when a community member registers with a domain map and contains the person's profile information. This information can vary, but is generally the person's name, email address, place of work and such. In addition, a value matrix is attached to the human resource and collects data about the person's actions while using the domain map. Thus, other users can browse human resources as well as knowledge resources and find who contributes the most to the knowledge-work ecology and in what ways. This will be discussed further in the next chapter.

Chapter 4

Value Matrix Object

"Strive not to be a success, but rather to be of value. "

– Albert Einstein

In this chapter, we will provide an overview of the general idea and the initial design of the Value Matrix Object and its charactertistics. The first section of this chapter is dedicated to the general overview of the VMO, while we later discuss a set of possible dimensions in the value matrix as well as how data for these dimensions can be collected. Further, we discuss how a value matrix can help us determining the value of a resource with respect to a context. We define a "resource" as both knowledge resources and users. Then, we examine how a value matrix can be attached to people and how it can be used in conjecture with the value matrix of knowledge resources. Lastly, we talk briefly about the benefits of combining the Value Matrix Object with the Domain Map Object.

4.1 General Overview

We established in Chapter 1 that we can use the Value Matrix Object together with the Domain Map Object for systemic innovation and create new knowledge work systems and ecologies. The idea is that a value matrix is associated with a resource and accumulates relevant information for evaluating the value of the resource it is associated with [55][59]. By collecting data this way, the system can evaluate the value of each resource and return the best answers for different contexts and queries. This is what we call *federating value information*, which is a key point of the Value Matrix Object.

Consider the following scenario: A novice user is looking for the best quality resources about a subject that is within his knowledge level. Attaching a value matrix to all resources in a system enables such queries. What will happen is that the system looks at each resource's value matrix, use the data within them to evaluate the resource and returns to the user the best results for his query. Another scenario might be that a user wants to find all articles that another given user has evaluated as groundbreaking research. The Value Matrix Object can do this as well.

The Value Matrix Object has two main application purposes. It should be able to distinguish the "good" resources from the "bad" resources so that, for example, good and relevant research is brought forward. However, what is considered "bad" today might not be so tomorrow or what is "good" and useful today could be outdated in the future. Hence, the Value Matrix Object must enable positive change. The system must not let only the majority's opinion count.

Figure 4.1: A value matrix is attached to each resource

In addition, a value matrix is associated with each user. This is done in order to evaluate the contribution of each user, which there are two reasons for doing. Firstly, users can be rewarded, both within the system by, for example, giving their contributions a higher weight, or outside the system where a high contribution value might be positive for the user's career. Thus, the VMO is an enabler of both intrinsic or extrinsic rewards for contributors, which naturally solves the issue of having an incentive to contribute, an issue that we discussed in Chapter 2 Section 2.4.5. Secondly, user contribution is evaluated in order to minimize the effect from those who deliberately damage the integrity of the system. In other words, the VMO provides the affordances needed to balance a knowledge work ecology by encouraging the right behavior. Naturally, the VMO also supports suitable practices by providing the means to reward those who contribute to any knowledge work activity, not just volume production.

We recall from Chapter 1 that the Value Matrix Object, just like the Domain Map Object, acts as a boundary object that enables communication between two or more communities. In this role, the VMO can be seen as a communication channel that lets the communities it connects to collaborate on its design and further development, which naturally is key to enable systemic innovation.

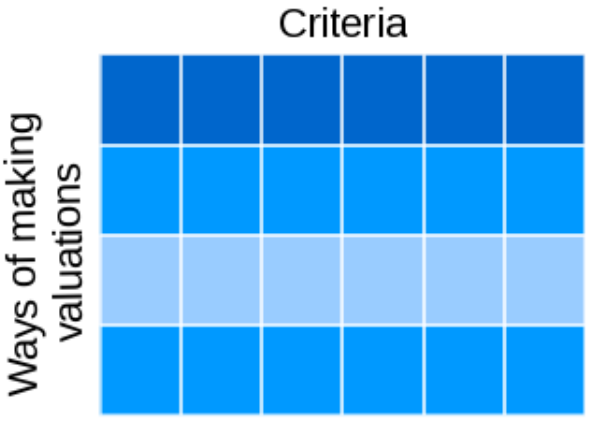

Figure 4.2: Conceptual view of the Value Matrix Object

Conceptually, we can view the Value Matrix Object as a matrix where the rows are criteria and the columns are ways of making valuations [55][59]. It is obvious that criteria for evaluating

the resources must be different from evaluating the users. The reason for this is of course that the two have different traits. The columns, on the other hand, can be the same because different criteria can be evaluated using the same valuation technique such as expert judgements or popular vote. It is also possible that some criteria do not use all of the possible valuations.

The value of a resource is obviously context-sensitive. That is why we use the Domain Map Object in conjunction with the Value Matrix Object. We can define context as a user's profile, i.e. his or her value matrix, and the user's position in the domain map. This means that all data that has been gathered for a user's value matrix determines a set of preferences that together with the position on the map is used when finding resources. In addition, the domain map a provides control panel with some functionality that allows the user to fine-tune how the value of each resource is computed. The result is that the user is able to dictate which resources are brought forward. See Figure 4.3 for a visual example.

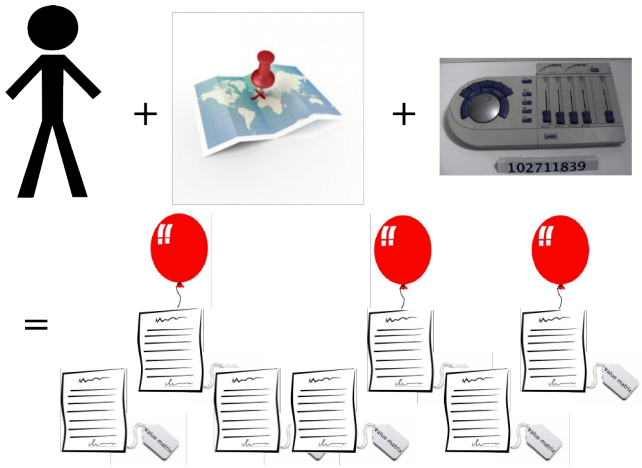

Figure 4.3: How resources are brought forward given a context

4.2 Affordances and Practices

The following is a subset of possible affordances that the Value Matrix Object provides [59]:

- **Customizing the value matrix**. Each community can customize the VMO to choose the values they want to support.
- **Assigning value matrices to resources**. This pattern is an addition to the Domain Map Object pattern "*putting resources on the map*", as a resource's visibility on the map is now relative to its value instead of all resources being equal.
- **Logging contributions**. Not only are the contributions to the value of a knowledge resource themselves valuable, but also the identity and value of the community members who provide the contributions. That information is also a relevant part of the evaluation record of a knowledge resource.
- **Logging events**. The VMO enables the logging of all activities, such as visits, citations and similar, that may be relevant for evaluating the resources.

- **Querying value matrices**. The VMO provides functions and algorithms through affordances that take advantage of the collected information for querying resources.

The above patterns enables a set of practices that the Value Matrix Object facilitates [59]:

- **Precise query**. A key role of the VMO is to collect and provide all information that is related to the value of resources. Naturally, this is important in order to be able to distinguish a subset of "right" resources from a large set of competing ones. For this to work, the system must contain a sufficient amount of relevant information, which is exactly what the VMO accumulates.
- **Creative reputation management**. The VMO records and rewards all contributions, aiming to create a good systemic ecology. A community member's profile reflects his reputation in a community. Thus, the VMO enables, for example, a promotion committee to evaluate candidates based on their profiles.
- **Creative query strategies**. With the right affordances, the VMO enables the use of filters and sorts in order to retrieve the resources we are interested in. For example, we might want to find all resources that a respected community member has ranked as outstanding.
- **Creative remuneration**. An important part of the VMO is to stimulate and encourage quality over quantity.
- **Crowdsourcing without overloading**. One core philosophy is that everyone should be given a voice and a chance to contribute, students and professors alike. The VMO supports this since the value of contribution is related to each person's profile and not his status in society.

4.3 Value Matrix Object on Knowledge Resources

In traditional Information Retrieval, IR, relevance is usually the deciding criterion when retrieving documents for a query [72]. Relevance in this context denotes how well a retrieved document or set of documents meets the information need of a user. However, traditional IR does not take into account the quality of the documents, only that the search terms in the query are present in the query result. With document quality we mean for example the accuracy, completeness, usefulness and understandability of the document content. Since relevance is obviously not a sufficient criteria for evaluating resource in the VMO, we introduce two additional criteria to help us with this issue. These are *quality* and *importance*. In other words, we now have a set of criteria; quality, relevance and importance (QRI), to help us evaluate the value of a resource. In this section, we will discuss these concepts and how they are used in conjunction with the Value Matrix Object. Additionally, we will present a set of minor dimensions; *recency*, *knowledge*, *keywords*.

4.3.1 Quality

Quality, with respect to resource content, reflects the intrinsic value of the information [56]. If the information contained within a resource is unreliable, incomplete or impossible to understand it is worthless. This holds true even if the information is also relevant or important.

Information quality is generally regarded as a multi-dimensional concept, but the characteristics may vary [60]. In 2005, Shirlee-Ann Knight and Janice Burn carried out an analysis on

how to determine the quality of Internet resources [61]. In the study, three main dimensions of quality are identified: Reliability, availability and relevancy. In addition, the study claims that the quality of information is a compound criterion of several specific characteristics. Table 4.1 shows the dimensions that Knight and Burn found was most commonly used to determine information quality [61].

Sub-characteristics of Quality in the Value Matrix Object

Obviously, not all of the dimensions from table 4.1 are applicable to the Value Matrix Object. Additionally, since several of them are very closely related to each other it is possible to merge them together into a single characteristic. We start by eliminating the following dimensions: Consistency, Security, Concise, Amount of Data, Navigation, Useful and Efficiency. The reason that we eliminate these is that we believe they are not relevant for the Value Matrix Object. Next, we eliminate Relevancy and Reputation. The former because we view it as a major dimension on its own. Reputation, on the other hand, we think of as a meta-dimension of quality instead of a sub-characteristic. That is, we compute a reputation score for a resource based on provided data from users, such as ratings and opinions, which inherently is an indicator of the quality of that resource. Algorithms that can be used to compute reputation scores was covered in Chapter 2 Section 2.4.3.

After we have eliminated the aforementioned sub-characteristics, we are left with Accuracy, Completeness, Reliability, Understandability, Timeliness, Accessasability, Availability, Objectivity, Believability, Useability and Value-Added. We mentioned that some of these are closely related and can with benefit be merged together. Table 4.2 shows which dimensions we want to use for sub-characteristics of quality in VMO, together with which dimensions from table 4.1 they are made up of and lastly, their definitions.

Manual Collection of Quality Data

With manual data collection we mean data that is collected through active user participation. Classic examples include filling out forms or schemas that a website presents to its users. However, it has become increasingly popular to ask users to express their feelings or opinions about an object, especially on social networks and e-commerce sites. There are many ways of doing this, such as like-buttons, rating systems, ranking systems and others.

Many frameworks that determine the information quality of a resource, looks at the information from the users point of view. This is supported by Strong and Wang who claim that the quality of information cannot be determined without involving the people who use it [102][101]. Naturally, we also intend to involve users as a part of evaluating the quality of resources by letting them actively provide data that is collected through VMO.

We can collect manual data in various ways. The most obvious way is to expose functionality to let users rate each of the sub-characteristics we discussed from 1-10. The range of the interval is of course debatable. Naturally, all community members have the ability to provide ratings, but the weight or impact of each individual rating may vary according to the rater's profile and value matrix. That is, the ratings of a highly respected and knowledgeable community member may carry more weight than ratings provided by fresh members. However, as fresh members increase in stature, as reflected by their value matrix, their ratings will dynamically increase in weight. In order to compute an overall quality score from ratings, a reputation engine is involved. It collects all ratings for a resource, taking into account their weights, and uses an algorithm that produces a quality score from the ratings.

In addition to providing ratings, some users will be given the possibility to provide a special distinction, "Michelin star", like those given to restaurants in the Michelin Guide, to resources

Common Dimensions of Information Quality	
Dimension	Definition
Accuracy	The extent to which data are correct, reliable and certified free of error
Consistency	The extent to which information is presented in the same format and compatible with previous data
Security	The extent to which access to information is restricted appropriately to maintain its security
Timeliness	The extent to which the information is sufficiently up-to-date for the task at hand
Completeness	The extent to which information is not missing and is of sufficient breadth and depth for the task at hand
Concise	The extent to which information is compactly represented without being overwhelming
Reliability	The extent to which information is correct and reliable
Accessability	The extent to which information is available, or easily and quickly retrievable
Availablity	The extent to which information is physically accessible
Objectivity	The extent to which information is unbiased, unprejudiced and impartial
Relevancy	The extent to which information is applicable and helpful for the task at hand
Useability	The extent to which information is clear and easily used
Understandability	The extent to which data are clear without ambiguity and easily comprehended
Amount of Data	The extent to which the quantity or volume of available data is appropriate
Believability	The extent to which information is regarded as true and credible
Navigation	The extent to which data are easily found and linked to
Reputation	The extent to which information is highly regarded in terms of source or content
Useful	The extent to which information is applicable and helpful for the task at hand
Efficiency	The extent to which data are able to quickly meet the information needs for the task at hand
Value-Added	The extent to which information is beneficial, provides advantages from its use

Table 4.1: The most common dimensions of information quality

The Sub-characteristics of Quality in VMO		
Characteristic	Made up of	Definition
Reliability	Accuracy, Completeness, Reliability, Understandability	The extent to which the information is reliable, correct and free of error, complete and easy to understand.
Timeliness	Timeliness	The extent to which the information is not outdated.
Availability	Accessability, Availability	The extent to which the information is accessable and retrievable.
Believability	Objectivity, Believability	The extent to which the information is unbiased and credible.
Useability	Useability, Value-Added	The extent to which the information is found clear and useful.

Table 4.2: The sub-characteristics of quality in VMO

they view as outstanding or remarkable. The purpose of the star is to allow for the separation of the very top quality knowledge resources from the rest. Each community is free to select who should be able to provide these stars, but it is recommended to only give this functionality to a limited number of people. Additionally, in order to avoid inflation, each member who is allowed to place stars should only have a handful of stars they can place. This limits the number of stars that can exist. A resource that has received a star will always be ranked higher than resources that do not have a star.

Automatic Collection of Quality Data

Automatic data collection differs from manual data collection in that user action is not required, but uses scripts, Internet cookies and other technologies to gather data instead. For example, collecting metadata from a web page is considered to be automatic data collection. Web crawlers are heavily used in such activities and play an important part in search engines [72].

There are many methods we can use to collect various data automatically. We will here take a look at a few of those methods. First, when a new resource is added to a domain map, it is possible to look at the author's previous work in order to determine a base rate quality score. If the author has published generally good quality work in the past, then we can assume that the new resource is also of pretty good quality. Obviously, this is not always true, but the quality of a resource will anyway be impacted by, for example, user ratings. Related to this method, it is also possible to look at what the contributor, i.e. the user who added the resource, has previously added. If that user has only added poor quality resources in the past, then it is likely that the next resource the user adds is also poor. Notice that there is a clear distinction between author and contributor. A community member who adds a resource to the community's domain map is the contributor, but not necessarily the author. Naturally, he can be both.

To summarize, we can look at either the value matrix of the author or contributor of a resource and their previous work or contributions in order to determine a base rate quality score for a newly added resource. The explained methods are best used to determine the base rate quality of a resource when it is first added to a domain map. However, this base rate may change dynamically as the value matrix of the author, contributor or previous work is updated.

Next, we can use natural language processing in order to detect the "mood" of citations within a knowledge resource [109]. That is, an author might cite an article to enhance a point within his

article. In this case, the author obviously thinks that the cited article is of high quality. Thus, we should update the cited article's value matrix in order to reflect this. The opposite scenario should also be considered. For example, an author might in his article discuss how things have been done poorly in the past. The articles he cites in relation to this should have their value matrices updated as well.

Related to the above method, it is possible to count how many times a resource has been cited and update its quality score based on that number. It goes without saying that self-citations, i.e. an author cites his own work, should be discounted. Both of these methods can be used to continuously update a resource's quality score throughout its lifetime.

Lastly in this section, we would like to discuss the impact the number of views a resource has should have on the quality score. It is not necessarily so that a resource with a high number of views contains high quality knowledge. It could also refer to the fact that the resource has just been added or that it is otherwise popular. By analogy, a popular movie does not necessarily win the Academy Award for best film. However, in our case the popularity of a resource might be an indication of quality, especially if it has a high number of views over a long period of time. Thus, a resource's views should be collected and stored for various time intervals, for example the number of views per week, per month, yearly and lifetime. The data can then be analyzed to see if there is a trend that could have an indication on quality.

Note that this is not an exhaustive taxonomy of ways to collect data, but rather a few examples of schemes that can be used. Also, keep in mind that it is possible to use any combination of the presented schemes to record data related to resource quality. Then, the collected data can be combined to compute a single number that reflects the quality of a resource. In this regard, each data collection scheme is a facet that can be combined with any number of other facets.

4.3.2 Relevance

As we have already mentioned, relevance in traditional IR refers to how well a retrieved set of information resources meets the information need of a user [72]. Relevance can also refer to how closely the retrieved results matches the query. The first definition is usually called user relevance and is subjective to the user, while the second definition is called topical relevance and can be judged objectively by experts [53]. Alternatively, topical relevance can also be computed at query time by the IR system.

In the context of the Value Matrix Object we have to define the term relevance to suit our needs because it is not a traditional search engine and terms from such systems are not applicable to the VMO. We recall that in a domain map we have a set of topics and a set of resources. Additionally, each resource is related to one or more topics. In this context, relevance is the relationship between a knowledge resource and a topic. In other words, if there is a relationship between the two elements or not. If there is a relationship, then the resource is relevant to the topic. We can ask *"does this resource belong in this topic?"*. If it does not, then it is not relevant for the topic and will not be displayed to the user and vice versa.

Collecting Relevance Information

Value Matrix Object relevance can be judged in three ways; by users, i.e. personal relevance, by experts or by the system itself. The judgment given by a user is collected by the system and is later used to compute a resource's relevance to a topic. For the purpose of computation, a reputation computation algorithm will be used. For example, it would be possible to use the average of ratings algorithm to compute a resource's relevance to a topic. However, it would be better to use a more complex algorithm that also takes into account the users' value matrix and the age of ratings so that positive (or negative) change is enabled.

An expert is also a user, but one who has been given the status of "expert". How users become experts will be further discussed in Section 4.4.2. The judgment of experts will also be collected and used to compute relevance, but the ratings from experts will be given a higher weight towards the total relevance score. As opposed to quality, the rating interval of relevance should be binary. We argue that either a resource is relevant or it is not.

Then, we have the judgments made by the system itself. The system can compare the likeness of a resource to other resources that have previously been deemed as relevant to a topic. If a resource is similar to another resource that is relevant to a given topic, we can assume that our new resource is also relevant to the same topic. Comparing resources has two uses. The first application is that it makes it possible for the system to give new resources an initial relevance score. In addition, it gives users the option to find resources that are similar to the one they are currently viewing, which is obviously useful in scenarios where they want to find more resources that are about the same specific subject. Comparing resources in this manner will be discussed further in Section 4.3.4.

Another possibility is to track whenever a user clicks on a resource to view its content. We can assume that if a resource is clicked on within a topic, then that resource is likely relevant to that topic as well. The number of clicks a resource has within a topic should count when computing the resource's relevance score, but with a lower weight than the ratings of users.

As with collection quality data, the discussed schemes are not an exhaustive list and can be expanded upon further research. It is also up to the community to find which schemes to use to collect relevance information.

4.3.3 Importance

Importance as a concept in traditional IR is almost non-existent. A reason for this might be that it is more subjective and philosophical than quality and relevance. Additionally, there is no viable framework that evaluates the importance of knowledge resources [68]. In the context of the Value Matrix Object, importance reflects the value of a knowledge resource from the point of view of a larger context. Alternatively, it can also reflect the value as seen by people who may possess knowledge about what a user might need [56]. It may not be clear why importance is an important characteristic of the VMO. The reason is that a knowledge resource can be important, but may not be judged relevant from the point of view of our present interests. However, this may change over time and the importance dimension suggests that a knowledge resource contains important information regardless of our current information need.

In a value matrix, importance can appear in three different situations or contexts. Importance can be personal to a user. In other words, a resource might not be viewed as important by the whole community, but may still be judged as important for a single user for whatever reasons. These resources will then appear in a personal list of important resources. Next, we have topical importance, which says something about how important a resource is to a topic. Even though importance is a subjective matter, we can apply the Wisdom of Crowds theory here. This theory is a counter-intuitive notion which implies that a group of people will repeatedly make better decisions than any decision made by a single member of the group [104]. An example of this is Francis Galton's observation that a crowd at a county fair could more accurately guess the weight of an ox than any single cattle expert [32]. If we apply this theory, we can say that the collective opinion of all users will be a good measure of how important a resource is. Naturally, this theory applies to all other dimensions where many people express their personal opinion as well. As with relevance, a resource's topical importance value van be judged by both normal users and experts.

The last context importance can appear in is universal importance. We define this as knowledge that is important regardless of the topic or topics a resource belongs to. Only experts are

allowed to judge the universal importance of a resource, simply because the system assumes that experts have more knowledge and experience than a normal user. However, this restriction can be regulated by each community as they customize their VMO.

Collecting Importance Information

A resource's value matrix collects importance information by receiving ratings provided by community members. As is the case with the quality dimension, the ratings are represented by integers in the range 1 to 10. These ratings primarily have an impact on the topical importance of resources. A reputation computation engine will be used to calculate an importance score for each resource based on the collected ratings.

A high rating indicates that the resource is important to the user who provided the rating. In order to avoid making the list of important resources unmanageable, the user can choose if he wants to add the resource to his list or not when providing a high rating.

In some situations it could be desirable to indicate that a resource contains important information without having a defined topical context. For example, a group of experts might want to raise the awareness of global warming to their entire community. This is where universal importance is applicable. Much like the quality dimension's "Michelin star", experts are allowed to mark a resource as universally important. These resources take precendence over the resources that only have normal user ratings when querying for important resources. These resources might even appear across topics.

All of the above schemes require manual action from users. It is possible to use document similarity techniques as well in order to automatically find an indication of importance for a resource. By comparing new resource to other resources that have already been judged as important, we can find this indication. In the next section, we will discuss document classification and similarity techniques.

4.3.4 Document Classification and Similarity

In information retrieval, document classification is used in order to assign a document to one or more classes. This may be done either manually by experts or automatically by using different algorithms. Automatic document classification can again be divided into two types: Supervised document classification and unsupervised document classification. The former uses some external mechanism, most commonly human feedback, that provides information on the correct classification for documents. The latter is also known as document clustering and the classification is done without any reference to external information [72]. Both methods are quite error prone due to the nature of the problem. That is, the classifications will not always be correct, but some algorithms will provide a lower error rate on average than others [72]. In fact, it was found by a study conducted by Caruana and Niculescu-Mizil that even if some algorithms perform better or worse than others on average, the models with poor average performance occasionally perform exceptionally well and vice versa [13]. This is due to the variability across the problems one wants to solve using a classification algorithm. In other words, there is no "right" algorithm to use in all situations.

The Value Matrix Object can use document classification and document clustering in various ways as we have already mentioned. In fact, manual document classification is performed when a user adds a new resource to a domain map. The user is then regarded as an expert for that resource and is allowed to make a judgment for which topic the resource is relevant to. By doing this, the resource is given an initial relevance score for the chosen topic. However, the newly added resource might belong to more than one topic. This problem can be solved in two ways. The first solution is to use a supervised document classification algorithm to see if the added

resource belongs to any other topics as well. We use a supervised algorithm because we already have a given set of topics that a resource can belong to. Examples of algorithms it is possible to use includes the Naive Bayes classifier, the k-nearest neighbours algorithm or support vector machines [72][28][19]. However, all automatic document classification algorithms have a flaw. They require a rather large document bank in order to be effective. That is, many knowledge resources must already be present in a domain map before such algorithms can reliably be used, though this is only a problem in the early lifetime of a map.

The second solution is to allow users, possibly experts only, to provide suggestions to what other topics a resource belongs to after the resource has been added to the system. Thus, while a user browses the system's resources he or she might find a resource that is relevant in another topic as well. The user can then mark that resource with the topic it should belong to. A resource's relevance to a topic is then finally controlled by the user community, where the system exposes the functionality to rate a resource's relevancy to a given topic. The system will then use these ratings to decide whether or not or to what degree the resource is relevant to that topic.

The supervised classifications algorithm has other uses in our system than just determining if a resource is relevant to a topic or not. For example, we could use the k-nearest neighbours, or the more time efficient ϵ-approximate nearest neighbors, to find resources that are similar [21][69]. By finding resources that are similar to each other, we can find all resources that are about a specific subject or to find resources that are similar to an important resource, thus indicating that the found resources are important themselves. In addition, the algorithm can be used to weed out duplicate or near-duplicate resources that might have been added over a map's lifetime.

4.3.5 Recency, Knowledge and Keywords

In addition to the QRI dimensions and characteristics there naturally exists some other, minor, dimensions that value matrices should contain and collect data about. In this section, we present three of them. Our goal is not to present a complete taxonomy of possible dimensions, so subject to further research there could be even more.

Recency

One of the minor dimensions is *Recency*. Whenever a resource is added to a domain map, its value matrix sets a timestamp that contains information about when the resource first added. This characteristic is obviously useful for situations where a user wants to sort resources ascending or descending using recency as the sorting criteria. Also, the recency dimension can be used to group resources by when they were added. This gives users the option to display, for example, only the resources that were added during the last week or the last month.

Additionally, a resource's value matrix can keep track of when it was last updated. This "time updated" characteristic serves some of the same purposes as the "time added" characteristic, but allow users to find the resources that were last updated. That is, resources that have been recently interacted with. Combined with the number of views a resource has, it is possible to find the resources which are popular right now in the community. Thus, discussion about these resources is encouraged. Because of the different uses they have, the two time characteristics should be kept separate in the value matrix.

Knowledge Level

The content of a knowledge resource requires some degree of experience from its readers so that they are able to understand it. For example, an article about Bayesian probability requires that its readers has some knowledge about mathematics, specifically in the realm of probability computation. A reader with only limited experience with mathematics, for example only through grade school, will most likely have a hard time understanding the article's content. Therefore, we introduce the dimension *knowledge level* to the Value Matrix Object. With this dimension, users are able to filter out the resources that are either to difficult to understand or to basic for the user's information need.

For this dimension to work, all resources must be given a knowledge level at creation time. This means that the user who adds a resource to a domain map must use his own judgment to give the resource a knowledge level between, for example, 1 and 10. Further, other users are allowed to give their opinion about the knowledge level as well, much like how quality and importance are rated. The collected opinion of the users is more likely to give a good picture of the resource's knowledge level than the opinion of a single user, cf. the Wisdom of Crowds theory [104]. Nevertheless, the initial knowledge level given is important.

In addition to the knowledge level given, each user can set a knowledge threshold for each topic when browsing the domain map in order to filter out resources that are either too complex or too simple. The reason that this threshold must be set for each topic is because a user can have more experience and therefore knowledge in, for example, mathematics than in geography. Thus, the user's threshold in the mathematics topic will be higher, filtering out simple resources, than in the geography topic.

Keywords

Keywords, also known as knowledge tags, are generally used on the Internet as a type of meta-information that describes some aspect of a knowledge resource. They add additional value, context, and meaning to the information [35]. In the past, keyword meta tags were used by web pages to provide information about what kind of content the particular web page provided. However, this practice was heavily abused by webmasters who began to use keywords that had nothing to do with the website content in order to rank high on queries that would normally not include that site and thus increase traffic [20][103]. Since search engines have stopped considering these types of meta tags, they are no longer used for this purpose [20][103].

Keywords have since then risen in popularity on social network sites such as Flickr and Last.fm. The latter site is a social network with focus on music. Here, keywords are added to artists' profile pages and the system then generates a list of similar artists based on their shared keywords [67]. In other words, the keywords are used to categorize content or create a loose connection between entities.

Mimicking this functionality, we can use keywords in the context of the value matrix to create a loose relationships between resources. Thus, we enable *keyword search* in the domain map. We can then find resources that have the same subset of keywords, and by extension talk about the same general subject. For example, a user might search for the keyword "reputation system" and as a result get back all resources that contain that keyword. It is also possible to use several keywords in a single query. Then, the resources that have the largest subset of the query keywords will rank the highest.

A resource's keywords can be added both manually and automatically. The user who adds a resource to the domain map has the option of providing a set of keywords. Then, users exploring a domain map can also add keywords to the resources they browse. In order to limit potential abuse and keyword redundancy, tag clouds can be created [65]. In such clouds, the most popular

keywords are emphasized. We can use this knowledge to consider, for example, only the top 10% of a resource's keywords in keyword searches.

In addition to, or instead of, letting users manually add keywords, the value matrix can automatically generate the keywords. For some resources, like research articles, keywords are already provided in the article content. Then, it is easy to extract the keywords and add them to that article's value matrix. However, it is also possible to use document classification and natural language processing to generate keywords based on the content of the resource if keywords are not readily available. That is, we can use this technology to find the general subjects of resources and use these as keywords.

4.4 Value Matrix Object on System Users

So far, we have discussed how the Value Matrix Object applies to knowledge resources, but only briefly touched upon value matrices attached to users. A user's value matrix records all actions taken by that user. The reason that we introduce the Value Matrix Object to users as well is because we want to change the way knowledge workers are evaluated. Today, they are mostly evaluated by their academic publishing volume alone. While spending time knowledge organization is not directly discouraged, it does not further one's career. In fact, today's conventional evaluation methods indirectly leads to cognitive overload because knowledge workers are encouraged to produce quantity over quality, which is exactly what we want to change.

Associating a value matrix with a user has many implications. One is that it is possible to evaluate a user's contribution to knowledge work using several criteria, and not just the number of publications. These criteria include contributing to the organization and systematization of knowledge as well as spending time on that outstanding article that leads to a breakthrough within a particular field. These criteria, or dimensions, will be discussed in more detail later. A user's value matrix can also be used to further that user's professional career. Much like a reference from a previous workplace, it could be possible to refer to one's value matrix as an indicator of experience and commitment to knowledge work.

From the perspective of within the system itself, a user's value matrix is used to compute a reputation score for the user. This reputation score impacts the weight of the ratings the user provides for resources. In other words, a user whose interest lies in contributing to knowledge organization, will have a higher influence on the value of resource than a user who tires to use the system for personal gain or otherwise damaging the system's integrity. Thus, unwanted behavior will negatively impact both the user's value matrix and possibly his professional career.

4.4.1 Evaluating User Contribution

Naturally, in order to compute a reputation score for users, their value matrix needs to collect some data. We do not present a complete taxonomy of possible user VMO dimensions, but rather a few suggestions. We split the dimensions we have identified into two categories; contribution to organization and author contribution.

Contribution to Organization Dimensions

We here present three dimensions. The first is number that is impacted by ratings a user gives to resources. We call this number *ratings score*. This number can be computed in either a simple or more complex way. That is, the simple way would be to just increment the number by 1 every time a user provides a rating. However, this would encourage spam. Therefore, a more complex scheme would be not only let the amount of ratings provided impact the score, but also

introduce a weighting system that values each rating with a number between 0 and 1 based on how the community agrees with the rating. For example, a user gives the rating 1 to a resource which all the other users in the community have given an 8. Intuitively, the value of the newly provided rating should not be 1 because it differs too much from the general consensus. This consensus can be find mathematically by for example using the multinomial Bayesian reputation scheme described in Chapter 2 Section 2.4.3.

There are many ways to compute the value of the provided rating, but we suggest the simple scheme of subtracting 0.1 from the maximum value of 1 for each whole number the provided rating differs from the consensus. For our example, the rating value would be $1 - (0.1 * (8 - 1)) = 0.3$. The users new ratings score would then be the sum of the old score and the weight value from the provided rating. Obviously, opinions may change over time and a resource's value matrix changes. Therefore, a user's ratings score must be continuously updated by recalculating the rating value for all old provided ratings at a regular interval, and not only incremented when a new rating is provided.

The second dimension is a number that is computed from ratings other users have given the user in question. We call this number *users' score*. These ratings are collected and computed much like how the individual resource ratings *quality* and *importance* are collected and computed. The user's value matrix collects these ratings and provided them to a reputation engine that computes a score that is the users' score dimension.

Finally, the last manual dimension we present is a number that is associated with the content, i.e. resources, the user has provided during his or her lifetime. We have given this number the name *content score*. As with ratings score, we could just increment the content score with a constant number every time the user adds a new resource to a domain map, but this also encourages spam. Thus, we need a scheme that gives a value to each resource that the user has provided and computes the content score based on those values. Remember that each resource has the QRI-dimensions that can be used for this purpose. However, *relevance* is context dependant and does not say anything about the actual resource content. Therefore, we will only use the Q and I dimensions for computing the value a resource gives to the content score. In other words, we can add together the reputation score of the Q and I dimensions for each resource the user has added to a domain map, and set the result as the user's content score. However, this number could quickly grow very large and essentially negate the impact of the other dimensions in the user value matrix. Therefore, we set the content score as the result of the average of all Q+I-scores from the resources the user has provided. This gives us the following formula:

$$contentscore = \frac{\sum_{r_i \epsilon R_u} Q_{r_i} + I_{r_i}}{|R_u|}$$

where R_u is the set of resources a user has added and r is a resource. Naturally, this is just one of many possible ways to compute the content score. In the end, it is up to each community to decide upon the algorithm they want to use for this purpose.

Author Contribution Dimensions

Currently, we have two author contribution dimensions. The first is a number that increases by the number of resources in the system the user has authored. This number is called *author score*. To compute the total author score we can use the same formula as for content score, but instead of using the set of all resources the user has added, R_u, we use the set of all resources the user has authored, A_u. Naturally, for such a scheme to work we have to ensure that a user and the author of an article is indeed the same person. While extremely difficult, and maybe impossible, to do with already written resources we can create a connection between a user and

a resource by having authors add their unique value matrix ID to the resources they produce. These IDs would be generated upon registering with a domain map, and then attached to the registered user's value matrix. Then, by adding this ID to new resources the system can identify the author uniquely and give him or her credit for writing the article.

The second automatic dimension is a number that is incremented by the number of times a user's resources are referenced in other users' resources. This dimension we call *reference score*. Obviously, this dimension is dependent on the above dimension to work. The way we calculate the reference score of a user is by computing the average QI-score of the resources that reference the work of the user.

In the end, the dimensions from both categories can be added together to give a picture of how "valuable" a user currently is. This score can also be viewed as the user's reputation. Obviously, the higher this score is the more "valuable" the user is. It is important to note that the two dimensions categories can interesting on their own as well. We can, for example, be interested in the top organization contributors or the authors who publish the best work.

4.4.2 Becoming an Expert

By default, a system that deploys the VMO should make a distinction between a normal user and an expert. An expert has more functionality available, which has been discussed earlier in this chapter. However, we have not talked about how a normal user becomes an expert. Basically, there are two ways to achieve this status. First and foremost, the owners of a domain map may assign this status to any number of users they wish at any time. They also have the might to revoke an expert status from a user if he or she abuses his power. It is important to note that a map's owners are not necessarily experts themselves, but rather has as an administrator status which gives them permission to give and revoke expert statuses. Other than that particular power, the administrators are treated as normal users.

The other way a user can become an expert is to participate in a domain map's growth and evolution. As discussed earlier, every time a user makes a contribution, the user is given points that are recorded in the user's value matrix. When a user has accumulated enough points, the user may be rewarded with the expert status. The threshold that needs to be reaches is set by the administrators so that they can either increase or decrease the difficulty of becoming an expert. However, there are problems with this approach. The most glaring one is that it allows everyone to become an expert, which is clearly not wanted as it makes the entire expert status redundant. It also encourages unfair ratings and spam because the faster a user gains points, the faster he achieves expert status.

Another approach is to use ideas from reputation flow models which were discussed in Chapter 2 Section 2.4.3. Primarily, we can have a constant reputation weight for all users, and that the reputation is distributed among them. Then, the system can award, for example, the top 10% of the users with an expert status. Another approach could be to employ ideas from Slashdot's reputation systems discussed in Chapter 2 Section 2.4.4, where two layers of moderators create a healthy moderation environment. Regardless of the method of choice, it is imperative that ballot box stuffing and unfair ratings are discouraged in order to have a healthy reputation system and ecology among the users.

4.5 Value Matrix Object and Reputation System Issues

In Chapter 2 Section 2.4.5 we discussed common problems with reputation systems. We will in this section present how the nature of the Value Matrix Object solves these issues.

- **Incentives**. We recall that one of the biggest issues with traditional reputation systems is that there is often no incentives for raters to provide ratings since there is no direct reward given to them. The VMO solves this issue by rewarding all contributions, also providing ratings. Thus, users are inclined to contribute as it has a positive impact on their value matrix.
- **Positive rating bias**. While the VMO does not completely abolish this issue, we believe it drastically reduced the problem. There is no mechanism that hinders users from always giving resources the highest possible rating. However, since all actions are recorded in each user's value matrix, such behavior is discouraged. We believe that the value matrix will encourage users to provide honest ratings. There is also a question of motivation. Performing positive rating bias will have little to no personal gain in a system with sufficient users because a single person's ratings will be negligible in the whole picture.
- **Unfair ratings**. This issue is very similar to positive ratings bias and is solved in the same way by the VMO. Additionally, the VMO supports endogenous discounting of ratings by giving a low weight to ratings given by users who are perceived as biased or unfair by the system. Naturally, user value matrix plays a big role in this as it can be used to determine if a user is unfair based on previous ratings.
- **Change of identities**. People changing identities or making new profiles in order to reset previous behavior is a negligible with the VMO, since each new profile is given a new value matrix and it takes time to build a good profile and gain a good reputation within a community. Additionally, a community can control the access to their domain map by only letting people who are invited registering a profile. Furthermore, while making fake profiles could be possible, it cannot be used as a career enhancer in real life because of the mismatch between the profile and the real life identity of the profile's owner.
- **The time factor**. This is not a big issue since users primarily rate knowledge resources. Very rarely does the content of published knowledge work change over time, so the quality should remain the same. What can change over time is the importance of a knowledge resource. If this becomes a problem, then mechanisms associated with solving this issue, such as having an aging factor or similar on ratings, can be implemented. Alternatively, since the VMO records when ratings were given, users can be reminded at regular intervals to update their ratings.
- **Ballot box stuffing**. Since a value matrix only allows each user to rate each of a resource's dimensions once, ballot box stuffing will not exist. Providing a rating for a dimension more than once will simply just update the old rating.

4.6 Combining VMO and DMO

So far, we have talked about both the Domain Map Object, the Value Matrix Object and what kind of functionality they each enable on their own. However, we have not yet discussed the possibilities that arise when the two objects are combined.

Since the DMO is a map it naturally provides some positional context, i.e. where we are. Through the presence of topics, which can be viewed as areas on the map, each position reflects an area of interest. In other words, when a user browses the domain map, the user expresses interest in the content of the topics he visits. The user is then presented with the resources that are *relevant* to the current position or context. We remember that relevancy is one of the major dimensions of a resource's value matrix. However, this is not all. Each user also has his own

personal value matrix which reflects the user's interest areas, knowledge level and so on, all which are applicable to the context. Lastly, the user has a collection of affordances at his disposal which can be used to fine-tune search criteria. Together, these three sources of information presents a complete context that is used to evaluate which resources are presented to the user, thus we enable context-sensitive queries.

Furthermore, combining the two objects promote a good knowledge-work ecology through a game-like ecosystem which rewards contribution to the organization and federation of knowledge resources. That is, desired behavior should be rewarded either directly or indirectly. A system combining VMO and DMO can reward users directly by giving them more power within the system through working up a reputation based on their contribution. We have already discussed the perks of becoming an expert as well as being given a bigger voice in the form of a higher weight of your ratings. Indirectly, users are rewarded by gaining the trust and stature within the community. This can further be used as a possible career enhancer if a user can show his potential employers his level of reputation and status through his value matrix.

Chapter 5

Prototype Design

> "Design is directed toward human beings. To design is to solve human problems by identifying them and executing the best solution. "
>
> – Ivan Chermayeff

In this chapter, we will derive a design for the first prototype implementation of the Domain Map Object and Value Matrix Object. The first sections of this chapter deals with the goal of the prototype, assumptions made and the requirements for our solutions. Then, we present and discuss the general idea of the initial design, before we provide a high-level view of the components the prototype consists of. The last two sections are composed of a deeper look into the design of the client and the server components.

5.1 Goal

Our goal is to implement an initial prototype of the two objects presented in this thesis, the Domain Map Object and the Value Matrix Object, which were described in Chapter 3 and Chapter 4, respectively. Obviously, due to the nature of being a prototype, we will not implement all of the discussed functionality associated with the two objects, but rather a subset of them to visually show how we perceive the objects to work. The overall goal of the prototype implementation is to show how knowledge can be federated and organized through the Domain Map Object and the Value Matrix Object. A more detailed description of the general idea behind the prototype will be presented in Section 5.4.

5.2 Assumptions

In the design of this prototype we make a couple of important assumptions. First, we assume that all users have good intentions. We will not be implementing mechanisms that discourages spam or other actions that damage the system ecology. However, in future iterations of the prototype mechanisms that address these problems must be implemented.

Further, we assume that the location of the physical resource is constant. In other words, we do not take into account that the physical resource might be deleted from the server it is located on or otherwise moved. Thus, our encapsulated resource object is independent from the physical version. Also, we do not care if any changes to the content of the physical resource occur, seeing as these changes would be reflected in the resource's value matrix.

5.3 Requirements

There are some requirements that the prototype must fulfill. These reflect the purpose of the application.

- **No resources should ever be completely lost**. This means that any resource can eventually be found.
- **All user interactions with the prototype should be logged**. In order to best compute the value of resources given a context and a query everything a user does must be saved. This also helps keep the integrity of the application intact since, to the best of our ability, user opinions are never lost in the communication between client and server.
- **A user should always know where he is on the map**. It is important that a user always knows what it is that he is currently viewing.
- **Changes and additions to the domain map should be reflected to the users as quickly as possible**. When a user adds new resources to the map or even just provides his ratings, the prototype should immediately react and make the new additions visible.
- **The logical design should support future iterations and maintenance**. As we know, this is prototype is only initial and we want to support further iterations of it by making it possible to add new components to already existing interfaces.

Even if it is not desirable that users must wait for the prototype to complete computations, we have not taken into account the effectivity of the various algorithms in this prototype. However, this is something that needs to be considered in future iterations as the functionality of the prototype grows.

5.4 General Idea

In the design of the prototype, much of the inspiration for the look and feel was taken from the applications presented in Chapter 2. We have already talked about how Topic Maps and the Domain Map Object are conceptually similar, but both Debategraph and SpicyNodes were major sources of inspiration. Especially how users explore the domain in these two applications is prevalent in our design as well. Users explore the domain by clicking on nodes on the screen, revealing additional nodes and information. While this feature helps us organize nodes and encourage exploration, it does not help us eliminate the glut since there is no way to make a distinction between valuable and worthless nodes. In essence, all nodes in these applications are of equal value. That is why we have introduced the concept of reputation systems to our prototype. By continuously collecting data that is related to the value of a node, we can use a reputation computation engine to compute the node's value, thus enabling us to present only the most valuable nodes to the users. Naturally, this is where value matrices shine and are used. We also include user value matrices in our prototype, as we log all actions users take while exploring the domain map.

With the general concept decided upon, we needed to define the overlying scope of the prototype. Basically, we there are three different options. The prototype can either be a completely self-containing system, a system where the data is gathered within the system, but the knowledge resource themselves are located externally, or lastly a system where we would encapsulate the knowledge resources on their already existing locations. The latter design was quickly dismissed

because it would require that we communicated directly with all the servers that various knowledge resources are located on in order to be able to gather data about those resources, i.e. how many times they have been read, et cetera. This solution would also drastically reduce the type of data we could collect because it would depend on what the external location already collected.

If the prototype would be a completely self-containing system, users would have to upload the knowledge resources themselves to our server. This means that copies would be created and exist at two different places, the original location and on our server. We would then have the problem that changes to the original would not be reflected in the copy. In addition, having a copy of all possible knowledge resource would take up a lot of space on our server. On the other hand, we would be able to attach whatever overhead or metadata we need to each resource and it would allow users to upload resource that does not already exist on the web. However, also this design was disregarded because of the problems mentioned.

We settled on a system that is semi self-contained. What this means is that we do not store copies of the knowledge resources on our server, but rather a pointer to their original, external location. This design enables us to have exactly the same functionality as the self-containing system, but we save a lot of server space and since we no longer create copies, updates to the external resources are automatically reflected within our system as well. Although this design has a couple of drawbacks, they are rather minor. Drawbacks include that we cannot see what is happening to a knowledge resource outside our system and the resource must already exist somewhere on the web before it can be added to our domain map prototype.

5.4.1 Functionality

We will now talk about some of the functionality that we want to employ in the prototype. First, we need to make sure that users are able to add new objects to the domain map. Most often we want to add new knowledge resources, but new topics and resource comments can also be added. Before a new knowledge resource can be added to the domain map, the user must provide a link to the physical location of the resource, the name of the resource, its knowledge level, what topic it should belong to, and lastly who the author is. Once the resource is submitted, a value matrix is attached to it, and users can find it by browsing the domain map.

The main role of the Value Matrix Object in the prototype is to collect various information about resources, both knowledge resources and users. There are endless possibilities of data to collect and ways to collect this data, so we have chosen a handful. The most important dimensions are quality, importance and relevance, which are collected through user ratings. These QRI-dimensions are simplified in the prototype. We remember from Chapter 4 that especially the *quality* dimensions had many sub-dimensions. For the sake of simplicity, we do not consider these. Users just straight up rate resources' quality as a single dimension. In the same chapter, we also discussed various ways of collecting data for these dimensions. In the prototype, only user ratings are collected and used to compute a score for them.

The ratings that are collected and stored in a value matrix are used by a reputation computation engine that computes a reputation score for each of the mentioned dimensions. These scores are the basis for filtering out low value resources. We chose to use a rather simple average of ratings algorithm to compute the reputation scores because, after all, we want to see if concept behind the prototype is sound before spending a lot of time implementing a more sophisticated scheme.

We wanted to have some dimensions in addition the QRI ones, so we ended up using the recency, knowledge level and keyword dimensions as well. The value matrix keeps track of the time its resource was added to the map as well as the time when the resource was last interacted with. This way, resources that were added at the beginning of the domain map's lifetime, but are still interacted with, will still appear as a recent resource.

Briefly, we mentioned that when a user adds a new knowledge resource to the map he also sets its knowledge level. In the prototype, this is the only time information about the knowledge level is collected. Again, this is due to simplicity reasons and other ways of collecting knowledge level data might be considered in future iterations.

We use the concept of keywords in the context of the value matrix to create a loose relationship between resources. It enables the use of 'keyword search' through a control panel to find resources that have the same subset of keywords and display them on the domain map. For example, a user might search for the keyword "reputation system" and as a result get back all resources that contain that keyword. For simplicity, we have omitted the possibility of having several keywords in a single query. In this prototype, a resource's keywords are added manually by users, but it would also be possible to have keywords automatically generated from the resource content.

Furthermore, we need a way for users to interact with the domain map and affect which resources a shown. We discussed and experimented with using checkboxes to check off various criteria, knobs to fine-tune criteria and sliders that also fine-tuned criteria. In the end we chose sliders as they felt like the most natural and intuitive of the three assessed variants. We decided upon four different sliders that users can interact with; "knowledge", "recency", "quality" and "importance". The sliders can be seen in figure 5.1. At one point the relevance dimension had its own slider, but we came to the conclusion that it was not needed since a resource is either relevant to a topic or it is not. The "knowledge" slider operates in the interval [1, 10] and controls the lower bound of personal expertise required to understand the content of the resources. For example, a university professor might set this slider to "7" in order to only display the resources which have an expertise level of 7 and above because he is not interested in introductory articles in the topic he is browsing.

Next, we have the "recency" slider. This allows users to exclude resources that were added or updated after a given time. Currently, the slider has four levels; lifetime, 6 months, this month and this week. Obviously, manipulating this slider allows the user to only see recent resources or all resources if he wishes. The last two sliders are called "quality" and "importance". They both operate in the interval [1, 10] and obviously manipulates the level of quality or importance, respectively, a resources needs to have in order to be displayed. In other words, a user can, for example, choose to show only the very highest level quality resource, or everything that is not mediocre or poor.

Naturally, all sliders can be combined. For example, a user might want to look at all high quality resource that were added in the last month and requires a good level of expertise to be understood. By adjusting the sliders just right, this can be achieved.

Figure 5.1: Knowledge resource threshold sliders

Lastly, we need a way to query user value matrices as well. For this purpose we chose two dimensions that users can be queried over; *author contribution* and *organization contribution*. In figure 5.2 we see the two sliders that can be manipulated in order to filter users on the two dimensions. Both sliders work in the interval [1,10]. As with the resource sliders, both user dimension sliders can be combined in a query.

Every time a user makes a contribution, such as adding new resources, rating resources and other actions, that user will earn a number of points. All of these points goes towards the organization contribution dimension. Every user is put into a bracket in the interval [1,10]

based on his or her organization contribution score, and the corresponding slider filters out users bracket by bracket when manipulated.

Author contribution is very different from organization contribution. Points for this dimension are gained through the quality and importance scores for all resources a user is the author of. In the end, a user's author contribution score will be a number between 1 and 10, where 10 is the highest. A scheme for computing the author contribution score will be discussed in Chapter 6.

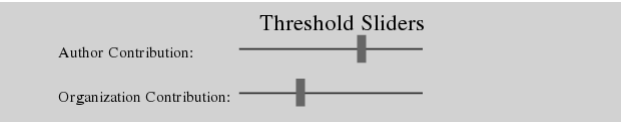

Figure 5.2: User resource threshold sliders

5.5 Overall High-level Prototype View

The prototype concists of both client and server functionality. Figure 5.3 shows a high-level overview of the system.

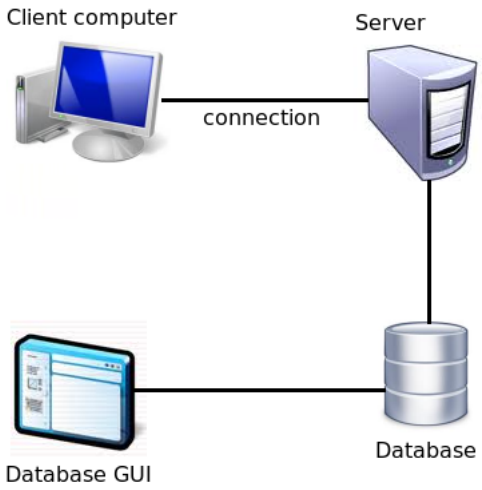

Figure 5.3: High-level overview of the system

In the figure we see the client computer to the top left. This entity loads the prototype and displays it through a browser so that the client's user can interact with the prototype. A selection of the interactions, those that generate data that needs to be stored, are communicated to the server. This entity is displayed to the top right in the figure. Then, all data is sent to the database, shown to the lower right, and stored. In addition to saving the data, the database is also active in retrieving data that is returned to the client. Lastly, we have the database GUI that helps human moderators to interact with the database's structure and data content.

5.6 The Client

The client is the main part of the prototype. It is displayed through a user's browser and is a visual representation of the Domain Map Object and its resources and topics. Additionally, it contains instances of the Value Matrix Object for the evaluation of resources. In this section, we will discuss the design of the client.

Figure 5.4 shows an abstract UML view of the prototype. Three classes have been simplified in this figure for size considerations. These are the GUI class, the Collections class and the Singletons class, who will all be discussed in more detail along with their respective UML diagrams later in this section. It is important to note that these three classes only provide a high-level overview of the design structure, and as such act as containers for other classes. They are not instantiated in the prototype.

The GUI class contains, as the name suggests, various classes that deal with GUI elements, such as menus and windows that users interact with, in order to manipulate what is shown on the screen. Further, the Collections class contains all classes related to topics, resources, both knowledge resources and users, as well as their respective value matrices. Lastly, we have the Singletons class. In the more detailed view of this class we find elements with vastly different functionality, but are collected in this abstract view under the same class because they are all singleton classes, and as such share a characteristic. What a singleton is and the details of these elements will be discussed later.

In figure 5.4 the main focus is on the DomainMapObject class. This class acts as the glue that holds all the classes in the prototype together. It contains most of the logic in the application and draws and displays the elements of the Collection class to the screen. Lastly, it acts as a relay between the other classes shown in the figure.

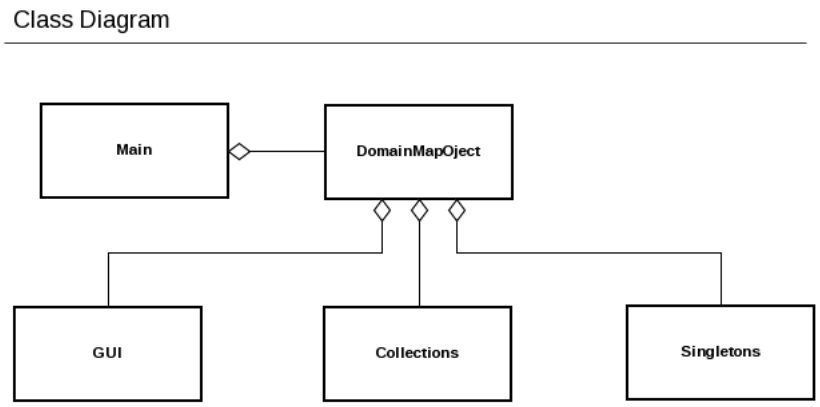

Figure 5.4: High level UML overview

We have used the concept of design patterns in the logical design of our prototype. A design pattern is a reusable generic solution to a common problem in object-oriented programming [33]. It is important to note that a design pattern is only a description or a template for how to solve a problem and not necessarily finished code that can be applied to any context. There are many reasons to use design patterns, but for us the most important reasons are the fact that it improves code readability for those who are familiar with design patterns and that the prototype becomes more robust with design patterns than if we had used an ad hoc design. Lastly, using design patterns means that our software becomes reusable and more easily maintainable.

5.6.1 GUI

The GUI is designed based on the principle of the State pattern because while the general look and feel remains the same, GUI functionality changes based on the application state. The intent of the State pattern is to allow an object to alter its behavior when its internal state changes. The object will appear to change its class [33]. The State pattern is typically used when an object's behavior depends on its state, and that its behavior must change at run-time depending on the state. Alternatively, State can be used in cases where we have large, multipart conditional statements that depend on the object's state. Then, we can use the State pattern to put each branch of the conditional in a separate class. Doing this, we can treat the object's state as an object that can vary independntly from other objects [33]. Figure 5.5 shows the structure of the State pattern.

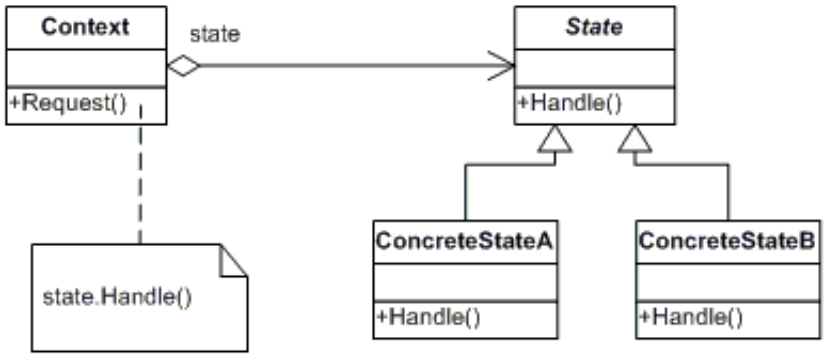

Figure 5.5: State pattern design structure

In the figure we can see three different types of classes. First, we have Context which defines the interface of interest to clients, and it is this class that a client interacts with so that it does not have to deal with the State objects directly. The Context class also has a reference to an instance of a ConcreteState subclass, which defines the current state. The Context's job is to delegate state-specific requests to the current ConcreteState. Normally it is also the Context's job to decide which state succeeds another and what requirements need to be fulfilled for a state change to occur. However, this job can also be delegated to the ConcreteState subclasses [33].

The next class we see is State, which defines an interface for encapsulating the behavior of the various states of the Context. Further, we have the ConcreteStates who subclass the State class. Each of the ConcreteStates implement the behavior associated with the state of the Context. A typical example is to have three subclasses that each implement the behavior of a TCP connection state, i.e. TCPEstablished, TCPListen and TCPClosed. The Context then starts at the TCPListen state before switching to TCPEstablished when a connection has been established. Lastly, TCPClosed is the final state that tears down and closes the established connection [33].

By using the State design pattern, we also introduce a set of positive consequences. We have already discussed how the State pattern localizes state-specific behavior and encapsulates states into interchangeable objects. This lets us easily add new states and transitions by defining new subclasses. An alternative to the State pattern is to use data values to define internal states and let Context check the data explicitly. However, then we would have long and likely similar conditional statements in the Context's implementation. Adding a new state is now no longer easy since we would likely need to change much of the existing implementation. In addition, code maintenance becomes increasingly complicated. We avoid this problem by using the State

Pattern. The trade-off is that instead of one compact class we end up with several small ones [33].

Furthermore, State objects prevent inconsistent internal states in the Context from occurring. This is because state transitions from the viewpoint of the Context are atomic. They happen by rebinding one variable instead of rebinding several [33]. Lastly, State objects can be shared between Contexts as long as the State objects have no instance variables. To be clear, the state they represent must be encoded entirely in their type for this to be possible.

In figure 5.6 we provide a more detailed view of the GUI class from figure 5.4. The classes to notice are GUIViewStateMachine, as well as AbstractGUIControlPanelView, AbstractGUISideView and their subclasses. The other classes in the figure are not that important and mostly provide auxiliary components and functionality to the aforementioned classes.

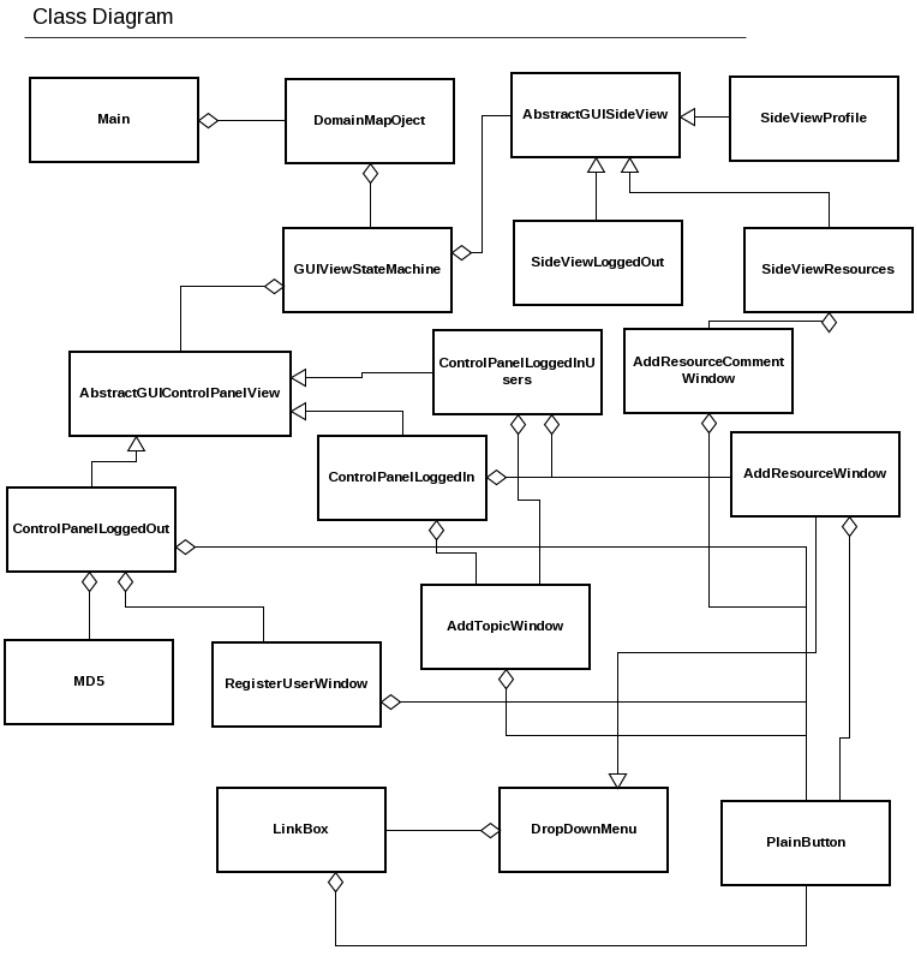

Figure 5.6: UML overview of the GUI classes

If we compare our UML diagram over the GUI classes to the structure of the State pattern, we see that GUIViewStateMachine is our Context. This class is responsible for our state transitions. Further, the two abstract classes AbstractGUIControlPanelView and AbstractGUISideView, are our States. We chose to have two State classes because they cover different areas of the GUI that

are independent of each other, the bottom of the screen and the far right of the screen respectively. Only having one State class covering these areas would complicate the implementation significantly and we would have to introduce extra variables to keep track of what combination of states we currently have.

Lastly, we have the State subclasses which implements our needed behavior for the states. ControlPanelLoggedOut, ControlPanelLoggedIn and ControlPaneLoggedInUsers all subclass the AbstractGUIControlPanelView class. The default state is ControlPanelLoggedOut. While in this state, the GUI offers the possibility to either register a new account or login with an existing account. The state changes to ControlPanelLoggedIn when a user has successfully logged in. This state allows users to provide new resources and new topics to the system, as well as adjust criteria for which knowledge resources are shown. Finally, transitions between ControlPanelLoggedIn and ControlPaneLoggedInUsers happen when a user clicks on a button that changes the view from showing knowledge resources to showing user resources instead, or vice versa. The ControlPaneLoggedInUsers state allows users to see the profiles of all users as well as adjust criteria to manipulate which user resources are shown.

The other State class, AbstractGUISideView, has three subclasses; SideViewLoggedOut, SideViewProfile and SideViewResources. SideViewLoggedOut is the default state. In this iteration of the prototype it does not provide any functionality. Rather, it only tells the user to log in to gain additional functionality. The state changes to SideViewProfile, which provides some information about the user's profile, whenever a user has logged in. Additionally, this state is transitioned to when a user clicks on a user resource. In future iterations, additional functionality to this state can be added. An extended piece of functionality can for example be letting users provide their professional opinion about other users. The last state, SideViewResources, is transitioned to from either of the two other states when a user clicks on a knowledge resource. This state provides information about the resource, as well as providing functionality that lets users interact with the resource, by letting them rate the QRI-dimensions, and add keywords among other things. An important note is that a user who is logged out is not allowed to interact with the resource, but the information is still provided.

Visual GUI Design

While designing the prototype, two iterations of the GUI were considered. The first iteration was a minimalistic design centered around clicking on buttons and hovering the mouse over objects to open windows that contained information and additional functionality. In figure 5.7 we see three screenshots, separated by black lines, from the old design.

In the top left we see the initial view of the first domain map. As we see, there are no panels or other GUI elements that provide further information, only three buttons, in addition to the topics, that users can click on. Clicking on one of the buttons brings up a window that obfuscates the other elements on the screen as shown in the top right of the figure. In this case, the button clicked was "*View Profile*", and thus profile information is shown in the window that has popped up. The last image at the bottom depicts a scenario where a user has clicked on a topic and holds his mouse over it afterwards. The idea was that a window would pop-up showing various criteria that could be checked off. Resources would then be filtered according to the criteria that were checked off. However, we found that the number of checkboxes would quickly get out of hand as more criteria were added. In addition, this method does also obfuscate the other elements, making it impossible to see what resources are shown before closing the pop-up window.

Because of the reasons stated above, as well as finding the overall look cluttered and uninspiring, we decided to redesign the GUI. The result from this second iteration can be seen in figure 5.8. The design is somewhat inspired by the Debategraph GUI which has two panels. One at the bottom of the screen that has ways of letting the user interact with the graph such as

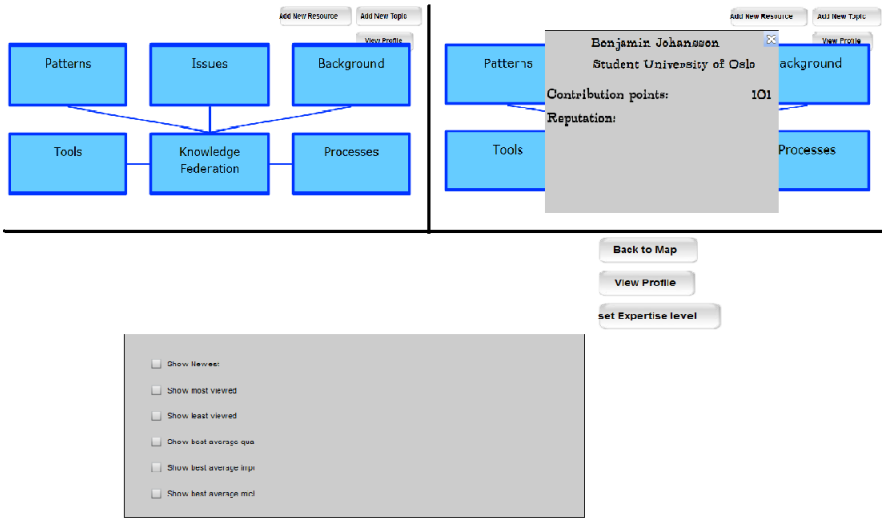

Figure 5.7: Old GUI look

adding new node or creating a new graph. The other panel is on the right side of the screen, and it provides more detailed information about the nodes that are currently being viewed. Furthermore, Debategraph encourages exploration through clicking on the nodes on the graph.

As we can see, the visual design has changed a lot. More specifically, the main GUI now consists of two panels, one at the bottom and one at the right side of the screen. In our UML diagram of the GUI part of the prototype, these panels correspond to AbstractGUIControlPanelView and AbstractGUISideView respectively. Notice that instead of a set of checkboxes, we have introduces the sliders we described earlier in this chapter. The view of the topics and resources, represented to squares and circles respectively, remained large the same after the GUI redesign. However, the redesign did spawn some additional functionality. First and foremost, the ability to leaf through resources in a topic by clicking on arrows. This functionality is rather important because it means that a resource will never be lost even if a topic contains hundreds of resources. Then, we also added two buttons, seen to the left of the resources in figure 5.8, that let users change between the knowledge resource view and the user resource view.

We have spared some of the pop-up windows, particularly the windows that let users register themselves or add new resources and topics to the map. We could have let these pop-ups subclass AbstractGUISideView instead so that they appeared on the right side of the screen, but this would require that the two panels were not independent of each other since the buttons that enable the pop-ups are located within the subclasses of AbstractGUIControlPanelView. Also, the actions that the pop-up windows perform are separate from map exploration activities, so we felt that keeping the pop-ups for these actions would not hinder users from exploring the domain map and its resources.

5.6.2 The Singletons

As mentioned, all the elements in the Singletons class in figure 5.4 are singletons. A singleton is a design pattern whose purpose is to ensure that a class only has one instance, and that there is a global access point to it [33]. It is important for some classes to have exactly one instance. For example, there is only one Prime Minister of Norway. Therefore, he or she is a singleton as there cannot be two Prime Ministers. Regardless of the personal identity of the active Prime

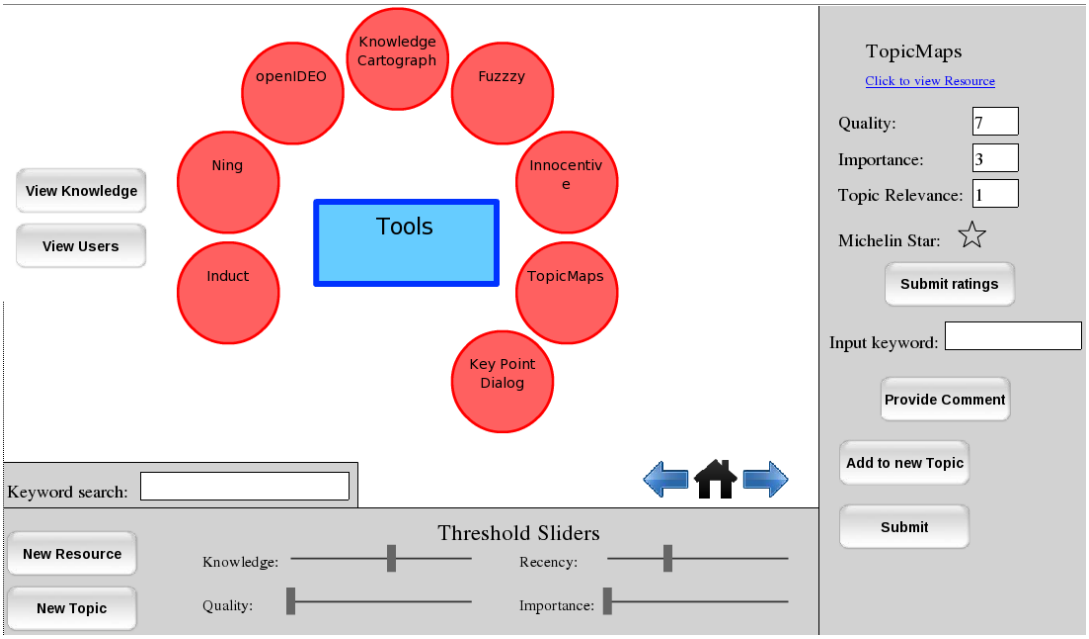

Figure 5.8: Current GUI look

Minister, the title "Prime Minister of Norway" is a global access point that identifies the person in office.

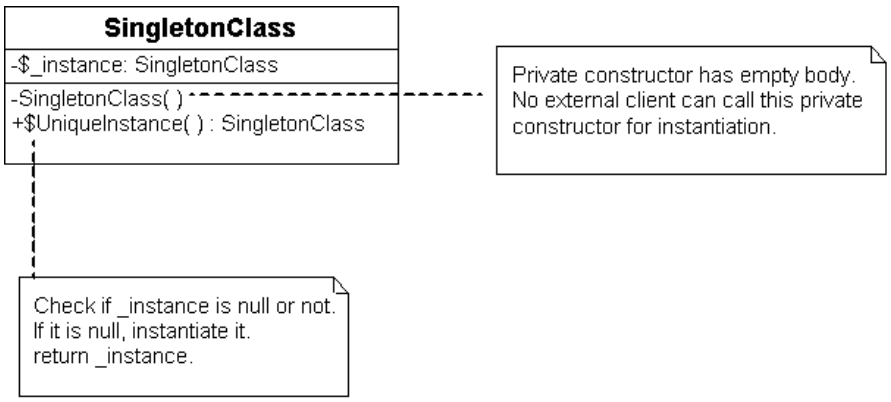

Figure 5.9: Singleton pattern design structure

An implementation of the Singleton pattern requires that the single instance and global access principles are satisfied. In other words, a singleton class needs a mechanism that allows a client to access the singleton class member without creating a class object every time. Also, the value of the class member must be persistent, i.e. it must be the same for all clients that access the singleton class. The implementation of the Singleton pattern is a class with a function, usually called *getInstance*, that creates a new instance of the class if one does not exists. If an instance exist, then a reference to the class object is returned. In addition, the implementation makes sure that an instance cannot be created in any other way by making the constructor private. Thus,

a client can only access a singleton instance through the getInstance function [33]. Figure 5.9 shows the structure of the Singleton pattern.

The Singleton pattern provides several benefits. The biggest is, as we have discussed, controlled access to a sole instance, which allows us to have strict control over how and when clients can access the class. Then, the patterns also avoids polluting the global name space with unnecessary global variables that store references to the singleton class instances. Last of the major benefits is the fact that the Singleton pattern permits lazy allocation and initialization, while global variables and static classes will always consume resources even before they are needed in the application [33]. Using the Singleton pattern has some drawbacks as well. The biggest is the fact that the pattern makes unit testing very difficult, as it introduces a global state into the application [39].

In figure 5.10 we provide a more detailed view of the Singleton class from figure 5.4.

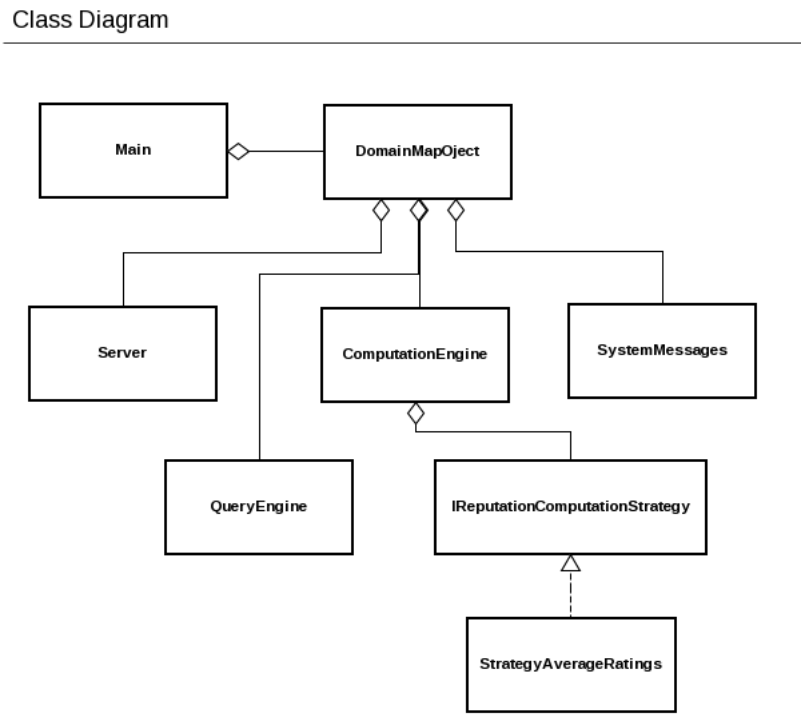

Figure 5.10: UML overview of the singleton classes

We have not mentioned this, but the DomainMapObject class is also a singleton. This allows us to have a global access point to the class instance, which is very handy in many situations, for example when we want to call a function in the SystemMessages class from somewhere in the class hierarchy. In addition, being a singleton we ensure that there is only one instance of the DomainMapObject class. Having more than one instance would not make sense conceptually and instantiating another copy would break the application.

In addition to the DomainMapObject class we see a few new classes in figure 5.10. The Server class and the SystemMessages class are neither very conceptually interesting nor complicated, so we will not spend a lot of time discussion them. Their purpose is to communicate data between the client and the server and display messages to the users through the client, respectively.

However, the ComputationEngine class and the QueryEngine class are not self-explanatory.

The Reputation Engine Design

Both the ComputationEngine class and the QueryEngine class are a part of a reputation engine. We remember from Chapter 4 that some of the dimensions in the Value Matrix Object will use a reputation computation engine in order compute their value scores. In this prototype it is the QRI dimensions, *Quality*, *Relevance* and *Importance*, that need to be evaluated by such an engine because the raw data for these dimensions are user ratings. If we go back to Chapter 2, we showed that there are many ways of computing an entity's reputation score, so it is up to the application designer to choose what algorithm to use. We have chosen to go with the Average of Ratings algorithm. Even if it is clearly inferior to, say, a Bayesian System, it is very easy to implement and get to work quickly, which is what we want in this prototype. However, in a future iteration of the prototype it is likely that we would like to change the method to a more sophisticated one, so we have designed this prototype so that the class StrategyAverageRatings can easily be exchanged with another class as long as that class also implements the IReputationComputationStrategy interface. In addition, we might want to change the computation method during runtime, i.e. for some type of data we use a Bayesian system while for another type we use a flow model. The design pattern that allow us to do this is known as the Strategy pattern.

The intent of the Strategy pattern is to define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it [33]. The pattern is generally used when many related classes differ only in their behavior and using it allows us to configure a set of runtime interchangeable classes that each perform one of many desired actions. In addition, we can use a strategy when we need different variants of an algorithm, just as is the case with our reputation engine. Figure 5.11 shows the structure of the Strategy pattern.

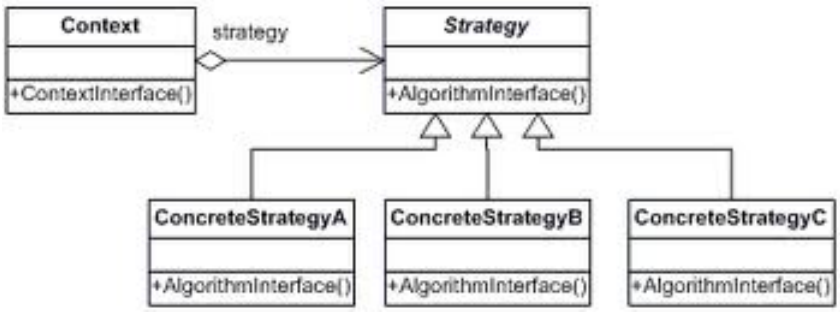

Figure 5.11: Strategy pattern design structure

The pattern works by letting Strategy and Context interact with each other to implement, or "activate", an algorithm. Then, a context forwards requests from its clients to its strategy, passing along all data that is required by the implemented algorithm. It is usually the clients that create and pass the ConcreteStrategy objects to the context, but after this exclusively interacts with the context. Clients usually have a variety of ConcreteStrategy classes to choose from and instantiate [33].

As with all design patterns, Strategy also has a set of advantages and drawbacks. The biggest advantage is what we already have discussed, the possibility of creating families of related algorithms that contexts can reuse. Alternatively, the Strategy pattern offers the possibility to let the client change between implementations at runtime depending on what kind of behavior is

needed. Lastly, we can use strategies as an alternative to subclassing. Imagine if we subclassed the Context class directly to give it different behaviors. Then, the behavior is hard-wired to the Context class and we mix the algorithm implementation with the Context's implementation, thus making it harder to understand, maintain, extend as well as being unable to vary the algorithm dynamically. In the end, we will have many different, but related classes whose only difference is the algorithm they implement. If we instead encapsulate the algorithms in separate Strategy classes we can vary the algorithm independently of its context, making them easier to switch, extend and understand [33].

There are also, as mentioned, some drawbacks with the Strategy pattern. Specifically that clients must be aware of the different strategies and how they differ in order to select the proper one. Thus, clients might be exposed to implementation issues. Further, there must be some communication overhead between the Strategy and the Context. Since all ConcreteStrategy classes share the same interface, it is likely that some of the simple algorithm implementations will not use all the information that is passed to them through the interface. This means that there might be parameters that are created and initialized, but are never used in the implementation [33].

If we compare the structure of the Strategy pattern to our UML diagram of the singletons, we see that the ComputationEngine class is our Context, while IReputationComputationStrategy is the interface that can be likened to the Strategy class in figure 5.11. In figure 5.10 we only have one class that can be compared to the ConcreteStrategy classes, which is the StrategyAverageRatings class. However, we could easily build a family of algorithms for the client, i.e. the DomainMapObject, to choose from. The algorithm family could include, for example, StrategyBayesianSystems, StrategyWilsonScoreInterval and so on, as long as they implement the same interface as StrategyAverageRatings.

The role of the ComputationEngine class is to take raw data from the database, which is essentially data from each resource's value matrix, and use it to compute a reputation score from each of the QRI-dimensions for knowledge resources. In addition, the class takes contribution data for users and computes their author contribution and organization contribution scores. We mentioned that the QueryEngine class is also a part of the reputation engine. Its role is to take the scores the ComputationEngine class has computed, and at query time decide which resources meet the filtering criteria a user has set. Naturally, these filtering criteria come from the threshold sliders we talked about earlier.

5.6.3 The Collections

If we take a more detailed look at the Collections class from figure 5.4 we find everything that is in figure 5.12. The most prominent classes here are the collection classes, TopicCollection, ResourceCollection, CommentCollection and UserCollection. When we talk about collections in computer science we mean grouping together a variable number of data items such as integers, strings or even custom objects. These items usually share some common trait or have some shared significance toward solving a given problem. Traditional examples of much used collections are lists, sets, bags, graphs and so on.

All of the collections in our prototype use an iterator in order for clients to access each collection's elements after they have been added to their respective collection. In short, an iterator is an object that traverses a container to access its elements. Obviously, there exists a design pattern for iterators. The intent of the Iterator pattern is to provide a way to access the elements of an aggregate object sequentially without exposing its internal representation. In addition, Iterator can be used to support multiple traversals of aggregate objects and to provide a uniform interface for traversing different aggregate structures, i.e. to support polymorphic iteration [33]. The structure of the Iterator pattern can be seen in figure 5.13.

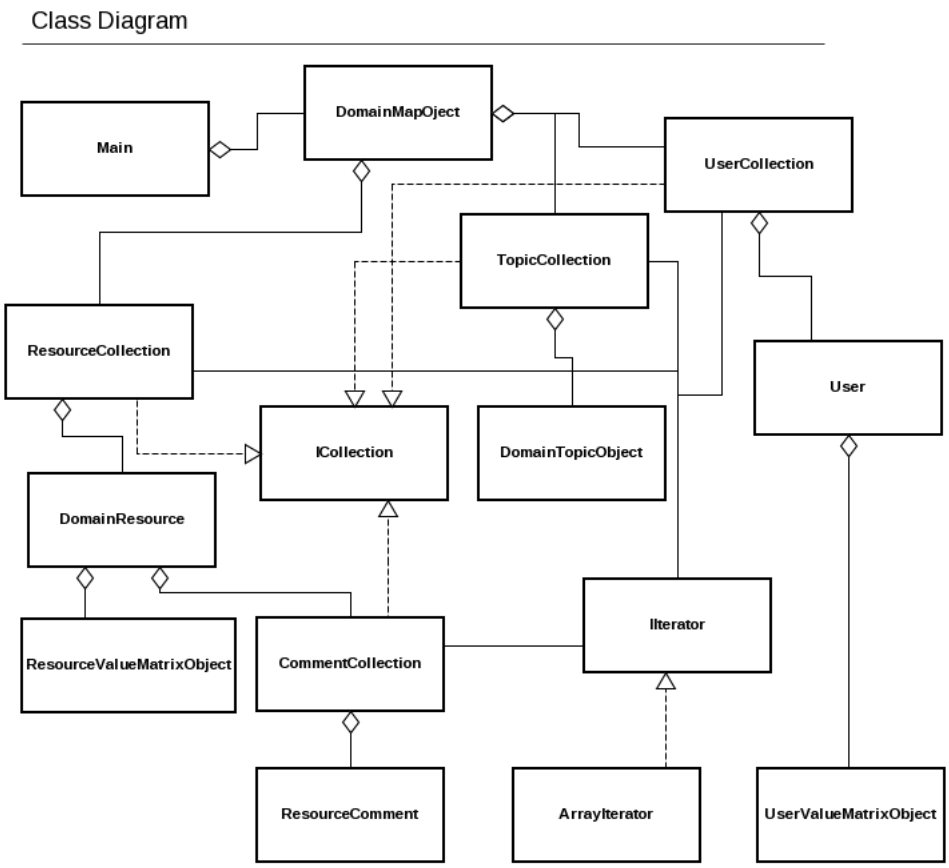

Figure 5.12: UML overview of the collection classes

In the structure figure we see two interfaces, Aggregate and Iterator, which are implemented by ConcreteAggregate and ConcreteIterator respectively. Iterator defines an interface for accessing and traversing elements, and Aggregate defines an interface for the creation of an iterator object. The job of the ConcreteIterator is to keep track of the current position in the traversal of the ConcreteAggregate and can compute the succeeding object in the traversal, while ConcreteAggregate creates and returns an instance of the proper ConcreteIterator [33].

The relationship between figure 5.13 and figure 5.12 might not be obvious. It goes without saying that Iterator and IIterator are the same and that ArrayIterator is an instance of ConcreteIterator. Further, we have that ICollection can be likened to Aggregate and that ResourceCollection, TopicCollection, CommentCollection and UserCollection are all ConcreteAggregates. Lastly, we have the Client class which can be compared the DomainMapObject class in that they both carry references to the iterators and the aggregates.

Using the Iterator pattern comes with some positive side-effects. Among other things, the patterns supports variation in the traversal of an aggregate. That is, Iterator makes it easy to change the traversal algorithm if necessary by replacing an iterator instance with another one. This is obviously useful if we want to have, for example, both prefix and postfix traversal of a tree. Furthermore, it is possible to define several iterator subclasses that support new traversals. Another side-effect is that the Iterator simplifies the Aggregate interface since the Iterator's

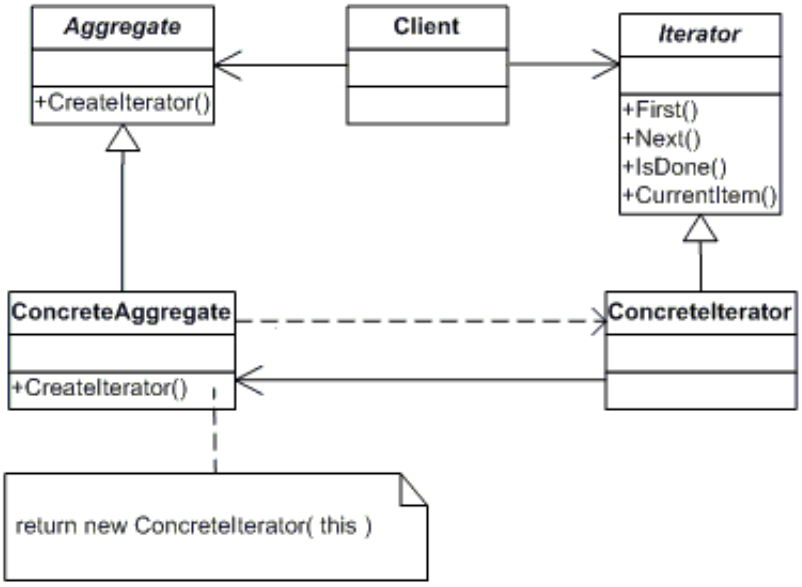

Figure 5.13: Iterator pattern design structure

traversal interface removes the need for a similar interface in Aggregate. Lastly, it is possible to have more than one traversal in progress at once because each iterator keeps track of its own traversal state.

In addition to the classes from figure 5.12 that we already discussed, we also have the DomainResource, DomainTopicObject, User, ResourceValueMatrixObject, UserValueMatrixObject and ResourceComment classes. In the following sections, these classes will be explained.

DomainTopicObject

A DomainTopicObject is very basic. The only data it contains is its name and an identification number. Instances of the DomainTopicObject class are contained within TopicCollection instances. When a TopicCollection is traversed, DomainTopicObject instances are returned to the DomainMapObject and drawn to the screen. When an entire collection has been traversed, the map has been drawn and users are able to explore it. Even though it may seem that the DomainResources are contained within DomainTopicObjects, they are not. For that to be the case, we would have to make a copy of a DomainResource for each DomainTopicObject it belongs to, which is both unnecessary and requires more computer resources. Instead, each DomainResource knows which DomainTopicObjects it belongs to.

DomainResource and User Objects

When a user clicks on a DomainTopicObject on the domain map, all DomainResources, which are knowledge resources, that are relevant to that topic will be drawn on the screen. In order to avoid cluttering the screen, as well as reducing the amount of information present at any one time, the amount of DomainResource visible at a time is limited. The DomainResources are sorted on their QRI score, which means that the DomainResources with the highest QRI score is displayed first and the rest are displayed in descending order. If the amount of DomainResources

belonging to a DomainTopicObject exceeds the display limit, the excess DomainResources can be viewed by clicking on a ”next“ button to leaf through the remaining DomainResources as described earlier in this chapter.

Since the QRI score is computed by quality * relevance * importance, only the DomainResources that are deemed relevant for the DomainTopicObject that is currently being viewed are displayed on the screen. The reason is obviously that if the relevance score is 0, then the entire QRI score is 0 as well.

While viewing a DomainResource, comments made by users about it are displayed in a similar fashion to how DomainResource are displayed around a DomainTopicObject. Each comment is contained within a ResourceComment object and are again collected within the CommentCollection class. These comments allow the users to express their opinion a little more than just giving each DomainResource a rating on the QRI dimensions. The ResourceComment enables users to use words to tell what they think about a DomainResource. For example, the comment functionality can be used to express that a DomainResource contains some vital knowledge about some subject. Another user can then use this information to make up his mind whether he should read the resource or not as an alternative to trusting the QRI ratings. Furthermore, other users can ”like“ or ”+1“ a comment to express whether the comment is helpful or not. The ResourceComments are sorted on how many ”likes“ they have received.

User objects are shown on the screen when the view changes from displaying knowledge resources to displaying user resources instead. Conceptually, User objects are similar to DomainResource objects. When clicked on, both objects display data on the right hand side of the screen. Naturally, the data is different for the two object types. While a DomainResource provides an external link to its physical location and exports functionality to rate it, a User object displays the name of the user, where he is from, his organization contribution, his author contribution and so on. Additionally, both sets of objects can be manipulated using a set of corresponding sliders. Lastly, both object types have a value matrix attached.

The Value Matrix Objects

As mentioned, a value matrix is attached to both types of resources. Conceptually, the value matrices for the respective resources are similar, but collect different data. The value matrix attached to a DomainResource contains raw data about the DomainResource's value, such as user ratings, keywords and so on. This data is used when the prototype computes the reputation scores for the quality, relevance and importance dimensions. In addition, a Michelin Star will positively impact the reputation score of both quality and importance by setting their individual scores to the maximum allowed value. The idea is that DomainResource that has been given a Michelin Star should be regarded as a ”cream of the crop“ resource.

The value matrix attached to a User object is in this prototype quite primitive. It contains the total amount of contribution points the user has accumulated and a reference to the DomainResources the user has authored. This data can be used to compute a score for the organization contribution and author contribution dimensions.

5.7 The Server

The server serves two main purposes. First and foremost it acts as a host for the prototype client application so that users can access it through their browsers. This part of the server has gone through a few design iterations. In the first iteration, users had to register and log in before being able to access the software prototype. We decided to move this functionality into the application itself in order to allow anonymous users to try it without having to register and

log in first. This method is more user-friendly and pretty standard for Internet applications. However, because of this change we had to restrict some functionality to only users who are logged in.

In addition to hosting the prototype application, we need a place to save the data that is accumulated as the prototype is used. The data we need to save is user information, knowledge resource information and data that is related to the value of both user and knowledge resources. Figure 5.14 shows the current database design. Obviously, as the prototype expands the type of data we need to save increases, thus requiring us to add more tables to the database eventually.

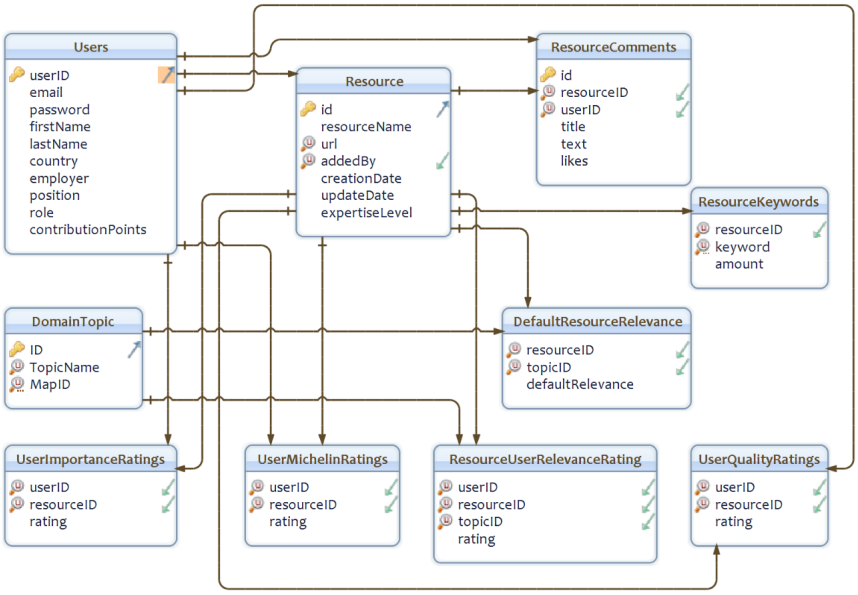

Figure 5.14: Database tables design

Notice that we have chosen to use a common relational database to store our data over a more modern object-oriented database or XML database. There are many reasons for choosing one of these modern database management systems over a relational database, but there is one reason that stands out. We are working with objects and an object-oriented database management system (ODBMS) would allow us to store objects originating from the client directly. Using a relational database management system, RDBMS, requires code that maps from objects to relational tables. Such code both takes up implementation time as well as increasing the cost of maintenance. Working with a RDBMS also requires that we work at a flat, primitive level, while an ODBMS allows us to work at any level. In addition, one of the functionalities that the Domain Map Object exports is the option to import or export knowledge, i.e. resources and their value matrices. Using an ODBMS naturally supports this functionality, while RDBMS does not.

However, importing and exporting knowledge is not a part of this initial prototype implementation, so that argument is not valid yet. Though, in a future iteration of the prototype, when the mentioned functionality is implemented, it would be feasible to think about changing to an object-oriented database to store our data in. In addition, we have previous experience using a relational database and the complexity of tables we need for this prototype is low and easy to set up and use. Therefore, we chose to use an RDBMS for this prototype.

Chapter 6

Prototype Implementation

> "A good idea is about ten percent implementation and hard work, and luck is 90 percent."
>
> – Guy Kawasaki

In this chapter, we will go through and explain the implementation of the design described in chapter 5. Since there is a lot of code, we will not present the complete code for all parts of the system, but snippets where this is appropriate instead.

Since the system is clearly split into a client and a server part as explained in chapter 5, this chapter will also be separated as such. First, we will address some general issues such as the choice of programming language and environment, before moving on to the implementation of the client. Lastly, the server side implementation of the prototype will be explained.

6.1 General Remarks

This implementation must be regarded as an initial prototype, and a proof of concept, only. Thus, it is by no means a finished product. We have focused on how the Value Matrix Object and the Domain Map Object can be combined and used in practice with regard to both knowledge resources and user resources, rather than producing immaculate source code. Therefore, it should be noted that a finished product would have additional functionality, more elegant approaches and solutions, and a more visually pleasing graphical user interface. However, we have tried to take this into account when writing code, so components can be changed out at a later stage, enhancing reusability and maintainability. In addition since this a client/server system, security measures for sending data over the network would have to be implemented, as well as stricter security for making new accounts, such as validating the users' email addresses. Also, it might be feasible to optimize some of the algorithms used so that as little computing resources as possible is used.

6.2 Programming Language and Environment

We have chosen to implement the client prototype using Adobe's Flash, which contains an object-oriented language called ActionScript that is in its third version [37]. Flash and ActionScript are used to create interactive applications, such as games, that can be integrated into web pages. ActionScript is a dialect of ECMAScript, which the more known JavaScript also is a dialect of.

Flash itself is an environment which, among other functionality, lets software developers create or import graphics and associate them with classes in ActionScript. In other words, Flash and ActionScript can together be used as a tool for the creation of rich Internet applications and quickly develop prototypes.

The reason we chose to use Flash and ActionScript is because we have previous experience with the environment and language as well as the possibility of quickly making high-fidelity prototypes. In addition, similar software such as DebateGraph and SpicyNodes, both which were discussed in Chapter 2, were created using Flash.

The server side of the system consists of three components. First, we use AmfPHP, a piece of third-party software that allows for easier communication between a client, in our case ActionScript, and PHP [64]. Without AmfPHP we would need one PHP-script for each unique function call between the client and the server. However, we use AmfPHP so that only one PHP-script on the server is needed because AmfPHP allows us to call specific functions within a script. The way AmfPHP works is that the client sends a request to a PHP-script on the server where AmfPHP is loaded. It parses the request, loads the requested service, calls it, and returns an answer accordingly. The AmfPHP layer is transparent to the end user.

Secondly, we have the PHP-script that AmfPHP calls requests on. This script contains a single PHP class which provides all the functions the client needs to either insert data into the database or fetch data that is returned to the client. The last component is the database, which is a MySQL database. This database type was chosen because it is well known, easy to work with and the fact that we have experience with the platform. Together with the MySQL database we have chosen to use a third-party GUI software component called phpMyAdmin. PhpMyAdmin is a tool written to handle the administration of MySQL over the World Wide Web and provides a visual interface that makes it easier for the database's administrators to maintain the database [98].

Figure 6.1 shows how each component is connected.

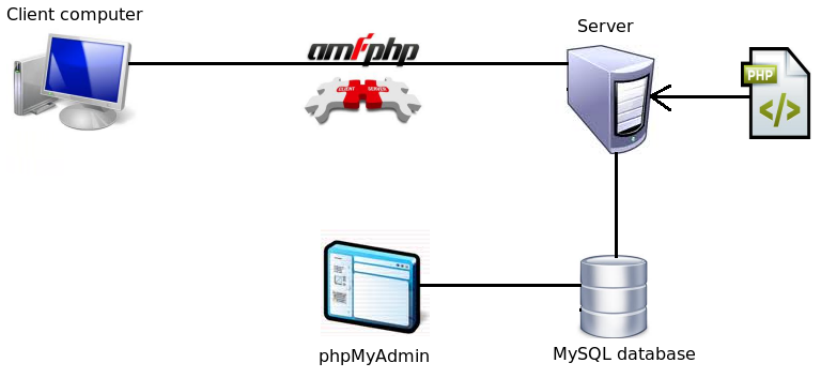

Figure 6.1: High level overview of the system

6.3 The Client

In this section, we will discuss various implementation details of the classes that the client consists of. We recall that in Chapter 5 that we split the client up in three categories; *GUI*, *Collections* and *Singletons*. This section will also be split as such, by first discussing the details of the GUI

before moving on to the Singletons and then the Collections. Finally, we will discuss the *Domain Map Object* class, which is the glue that holds the various parts of the prototype together.

However, first we must talk about the Flash environment and its library items. For this prototype we have created visual elements, which are added to a library, in the Flash environment. These elements can be linked to an ActionScript class so that they can be instantiated through code. Alternatively, they can be given an instance name so that they can be accessed in the ActionScript code with *this.instanceName*. This way of referencing graphical elements is often used when a graphical element contains one or more graphical elements, such as a window with several buttons. In order to be able to distinguish between class variables and these graphical elements, the convention is that class variables start with an underscore, while graphical elements are referenced using the *this* keyword. Figure 6.2 shows what the Flash library for our prototype looks like.

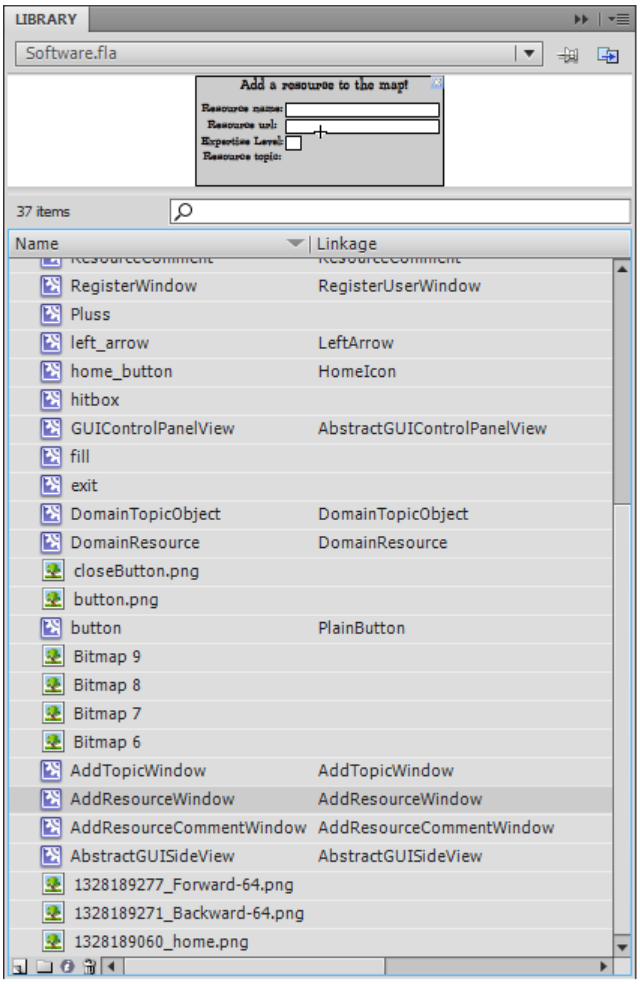

Figure 6.2: The system's graphical elements library

6.3.1 GUI

The design, both logical and visual, of the prototype GUI was discussed in Chapter 5 Section 5.6.1. Recall that we used the design pattern State so that we can easily make state transitions. It makes it possible for us to for example have a logged in state and a logged out state on the same GUI element. Then, the state we are in determines the functionality. The class *GUIViewStateMachine* can be likened to the Context class in the State pattern, and can be seen in listing 6.1. In this class there are a couple of things to notice. First, we have two variables that are references to other classes. These classes are our Strategies from the State pattern and are abstract classes that will be discussed later. Further, we have some static constants that are public. They are public because we want them to be accessible from other classes so that they can call, for example, *setState(CONTROLPANEL_IN_STATE)* in order to set the control panel's state to *logged in* from the default *logged out* state. By using the static constants we do not care about the actual string content of the constant. Changing the content of a constant will not have impact on other classes as long as the constant name remains the same. Lastly, we have the *setState* function. This function takes a string parameter, which is one of the static constants, and changes the state of either *_sideState* or *_controlPanelState* to the new state given by the parameter.

Listing 6.1: GUIViewStateMachine.as

```
1  public class GUIViewStateMachine extends MovieClip {
2    private var _sideState:AbstractGUISideView;
3    private var _controlPanelState:AbstractGUIControlPanelView;
4
5    public static const SIDE_OUT_STATE:String = "side_out_state";
6    public static const SIDE_PROFILE_STATE:String = "side_profile_state";
7    public static const SIDE_RESOURCE_STATE:String = "side_resource_state";
8    public static const CONTROLPANEL_IN_STATE:String = "controlpanel_in_state";
9    public static const CONTROLPANEL_OUT_STATE:String = "controlpanel_out_state";
10   public static const CONTROLPANEL_USERVIEW_STATE:String = "
         controlpanel_userview_state";
11
12   public function GUIViewStateMachine(){
13     setState(SIDE_OUT_STATE);
14     setState(CONTROLPANEL_OUT_STATE);
15   }
16
17   public function setState(stateDescription:String):void {
18     /* Set the correct state based on the stateDescription parameter */
19   }
20 }
```

Above, we briefly mentioned the two abstract classes *AbstractGUISideView* and *AbstractGUIControlPanelView*. They are the abstract classes for the bottom panel and the right side panel in figure 5.8, respectively. Normally, the State pattern has only one State class, but since we have two independent GUI panels we need to have two State classes as well. These abstract classes can be viewed as base classes for the ConcreteState classes and contain variables and functions that are common to all the ConcreteStates. *AbstractGUISideView* has three ConcreteStates, *SideViewLoggedOut*, *SideViewProfile* and *SideViewResource*. *AbstractGUIControlPanelView* has three as well in *ControlPanelLoggedOut*, *ControlPanelLoggedIn* and *ControlPanelLoggedInUsers*.

In listing 6.2, we see a part of the implementation of *AbstractGUIControlPanelView.as*. Since abstract classes are not a language feature in ActionScript 3, we must create a workaround [107]. This workaround consists of throwing errors in the abstract classes' functions that we want the

concrete classes to override. In addition, we throw an error if we try to instantiate the abstract class directly, thus we create a pseudo abstract class.

Listing 6.2: AbstractGUIControlPanelView.as

```
1  public dynamic class AbstractGUIControlPanelView extends MovieClip {
2
3    public function AbstractGUIControlPanelView() {
4      if(Object(this).constructor === AbstractGUIControlPanelView){
5        throw new Error("AbstractGUIControlPanelView must not be directly
             instantiated");
6      }
7    }
8
9    public function updateData(result:*):void {
10     throw new Error("updateData is an abstract method and must be overriden!");
11   }
12
13   /* Rest of this abstract class is filled with functions that are common to all
         states */
```

A ConcreteState that extends one of the two abstract classes has access to the defined functions in the abstract class it extends. In addition, it must override the functions in the abstract class that must be overridden. These functions are generally responsible for implementing the functionality that is unique to each ConcreteState. Obviously, the ConcreteStates are also allowed to define other functions as well. We have chosen not to show any code from the ConcreteStates, because it is mostly concerned about setting up text fields, buttons and other GUI elements and place them correctly on the appropriate GUI panel.

Finally, the GUI has many classes that represent buttons, windows and the like. We will not be going into the implementation details of these in the thesis proper. For more details, see Appendix A for access to the source code repository.

6.3.2 The Singletons

In Chapter 5 Section 5.6.2, we talked about the Singleton pattern and the design of the classes that used this pattern. We will now go into the implementation details of those classes, starting with the Server class before moving on to the classes that make up the Reputation Engine. Finally, we will briefly talk about the SystemMessages class.

The Server Class

In Section 6.2, we talked about how we use a third-party component called AmfPHP in order to enable easier communication between the client and the server. We will now discuss the client side code that is required to invoke server side scripts through AmfPHP. To accomplish this, we need a class that is responsible for the communication. Since the prototype has been programmed using ActionScript 3, the class has the *.as* extension and is called *Server.as*. Its shell can be seen in listing 6.3. A couple of things in the shell is related to the fact that the Server.as class is a singleton. These are the *_instance* variable, the *getInstance()* function and the *SingletonEnforcer* class. Thus, the rest of the variables and functions are related to the server connection. *initConnection* is responsible for the establishment of the connection itself, using the *_gateway* variable to connect to AmfPHP. Then, the *onfault* function is the common callback function that is invoked whenever a communication error occurs. Currently, it prints the error description to the console, but it would also be possible to print to a log file or similar.

Listing 6.3: The shell of Server.as

```
1  package{
2
3    public class Server {
4      private static var _instance:Server = null;
5
6      var _gateway:String = 'http://benjamij.at.ifi.uio.no/amfphp/Amfphp/';
7      protected var _connection:NetConnection;
8      protected var _isConnected:Boolean = false;
9
10     public function Server(e:SingletonEnforcer){
11       initConnection();
12     }
13
14     public static function getInstance():Server {
15       if(_instance==null){
16         _instance=new Server(new SingletonEnforcer());
17       }
18
19       return _instance;
20     }
21
22     protected function initConnection():void {
23       _connection = new NetConnection();
24       _connection.connect(_gateway);
25       _isConnected = true;
26     }
27
28     protected function onFault(responds:Object):void {
29       trace("There was a problem: " + responds.description);
30     }
31
32     /* The rest of this class is filled with request and response function pairs */
33   }
34 }
35
36 //We are outside the package so the following code can only be accessed internally
37 class SingletonEnforcer{
38   //this class enforces that the Server object is a singleton.
39 }
```

In addition to the shell, we also need a request and response function pair for each request going from the client to the server. Two such pairs are shown in listing 6.4. The two request functions, *loadResources* and *saveResource*, are called from elsewhere in the client and pushes a request to the server. We see that both functions are quite similar. First, they check if a connection has been established. Then, a Responder object is created. This object is responsible for making sure that the correct callback function is invoked when the server returns a response. On success, the function denoted by the first parameter is called, while the second function is called on failure. Notice that the *onFault* function is the default option when the server call fails. Lastly, the call itself is made through *_connection.call*. This function has two required parameters and n optional ones. The first required parameter denotes which script on the server and which function within that script should be called, while the second parameter takes the responder object that we created earlier. Lastly, the optional parameters allow us to send any number of variables to the server as parameters to the function we call. Notice that, except for pointing to its location in the *_gateway* variable, AmfPHP is transparent to us. It handles server requests and returns responses automatically. The only thing we need to do is connect to AmfPHP using the gateway variable and have the PHP script at a location specified by AmfPHP.

When a server call is successfully complete, the function specified by the first parameter in the responder object is invoked, allowing us to do something with the response from the server. In our examples in listing 6.4, *resultLoadResources* takes the data received from the server and sends it to the domain map object instance, which sets up the resources on the map. The other function, *resultSaveResource*, simply displays a message to the user saying, for example, that a new resource has successfully been added. Then, the client calls *loadResources* which fetches resources so that the newly created resource will eventually be shown on the map.

Later in this chapter, we will discuss the details of the two PHP-script functions, *getResource* and *saveResource*, that are called in listing 6.4.

Listing 6.4: Two request/response function pairs

```
1  public function loadResources():void {
2      if (!_isConnected) { this.initConnection(); }
3      var responder:Responder = new Responder(resultLoadResources, onFault);
4      _connection.call("VMOServer.getResources", responder);
5  }
6
7  protected function resultLoadResources(result:*):void {
8      DomainMapObject.getInstance().setUpResources(result);
9  }
10
11 public function saveResource(data:Array):void {
12     if (!_isConnected) { this.initConnection(); }
13     var responder:Responder = new Responder(resultSaveResource, onFault);
14     _connection.call("VMOServer.saveResource", responder, data);
15 }
16
17 protected function resultSaveResource(result:*):void {
18    DomainMapObject.getInstance().displaySystemMessage(result);
19    loadResources();
20 }
```

Reputation Engine

In Chapter 5 Section 5.6.2, we discussed the design of our reputation engine component. This component consists of the ComputationEngine class and the QueryEngine class. We recall that the ComputationEngine class uses the Strategy design pattern, so that we can easily switch between different computation methods at runtime if necessary. Listing 6.5 shows the functionality related to the Strategy pattern in the ComputationEngine class, which can be likened to the Context in the pattern. We notice two functions; *computeResourceReputation* and *computationStrategy*. The former computes the reputation scores for a resource from the raw user ratings by calling the *_currentStrategy.compute* with the appropriate parameters. Remember that by property of the Strategy pattern, the *_currentStrategy* can be exchanged at any time with another ConcreteStrategy implementation as long as that class implements the *IReputationComputationStrategy* interface. By calling the *computationStrategy*, we can achieve this exchange of computation strategy. We will discuss an implementation of the *IReputationComputationStrategy* interface later.

Listing 6.5: Strategy functionality in the ComputationEngine class

```
1  public class ComputationEngine {
2    private var _currentStrategy:IReputationComputationStrategy;
3
4    public function ComputationEngine() {
5    }
```

```
6
7   public function computeResourceReputation(resourceData:*):* {
8     var resourceReputationScores:Array = _currentStrategy.compute(resourceData["
          qualityRatings"], resourceData["importanceRatings"], resourceData["
          relevanceRatings"], resourceData["michelinRatings"], resourceData["
          defaultRelevanceRatings"]);
9
10    return resourceReputationScores;
11  }
12
13  public function set computationStrategy(strategy:IReputationComputationStrategy):
        void {
14    _currentStrategy = strategy;
15  }
16 }
```

The ComputationEngine class has two other important function as well. The first is *putUsersIntoOrgContributionBrackets*. This function is related to the computation of the value user organization contribution. In order to assign each user a number between 1 and 10, which is the interval the organization contribution slider operates in, we decided to put each user into a bracket. The user who has contributed the most to the organization of knowledge resources is automatically put into the highest bracket, which is bracket 10. Then, each other user is put into a bracket depending on their accumulated contribution points in relation to the highest contributor. In other words, a user who has more than 90% of the points of the highest contributor is also put into bracket 10, while a user who has between 40% and 50% of the points is put into bracket 5 and so on. Naturally, a user's bracket can change over time as he or she accumulates more points. The implementation of this algorithm can be seen in listing 6.6.

Listing 6.6: The putUsersIntoOrgContributionBrackets function

```
1 public function putUsersIntoOrgContributionBrackets(_userCollection:UserCollection)
      :void {
2   _userCollection.sortOnContribution();
3
4   var iter:IIterator = _userCollection.iterator();
5
6   var firstUser:* = iter.next();
7   var maxScore:int = firstUser.getUserContributionPoints();
8
9   firstUser.setOrganizationBracket(10);
10
11  while (iter.hasNext()) {
12    var nextUser:* = iter.next();
13
14    var percentageOfMax:int = (nextUser.getUserContributionPoints() / maxScore) *
          100;
15
16    if (percentageOfMax < 10) nextUser.setOrganizationBracket(1);
17    else if (percentageOfMax == 100)  nextUser.setOrganizationBracket(10);
18    else {
19      var bracket:int = int(percentageOfMax.toString().substr(0, 1)) + 1;
20      nextUser.setOrganizationBracket(bracket);
21    }
22  }
23 }
```

The second important function is *computeAuthorContribution*. This function is responsible for computing a score for the other major user value matrix dimension, namely author contribution. We ended up with using a slightly modified version of the formula presented in Chapter

4 Section 4.4.1 to compute an author contribution score. The modification is that we divide $quality + importance$ by 2 for each knowledge resource a user is the author of instead of having no division at all. We do this in order to get the resulting score in the interval [1,10], which is where the author contribution slider operates. Thus, the algorithm is: For each user, find all knowledge resources the user is the author of. Then, compute $\frac{quality+importance}{2}$ for each of the found resources and add the results. Lastly, divide this sum by the number of resources found, which gives us the average score. The implementation of this algorithm is shown in listing 6.7.

Listing 6.7: The computeAuthorContribution function

```
1  public function computeAuthorContribution(_userCollection:UserCollection,
       _resourceCollection:ResourceCollection):void {
2    var userIter:IIterator = _userCollection.iterator();
3
4    while (userIter.hasNext()) {
5      var nextUser:* = userIter.next();
6      var userName = nextUser.userName;
7      var authorSum:Number = 0;
8      var numAuthored:int = 0;
9      var resourceIter:IIterator = _resourceCollection.iterator();
10
11     while (resourceIter.hasNext()) {
12       var nextResource = resourceIter.next();
13
14       if (nextResource.author == userName) {
15         authorSum += (nextResource.qualityScore + nextResource.importanceScore) /
               2;
16         numAuthored += 1;
17       }
18     }
19
20     if(authorSum > 0) nextUser.setAuthorContribution(authorSum);
21   }
22 }
```

Naturally, we need some need some Strategy interface and at least one ConcreteStrategy to complete the Strategy pattern. Both of these, *IReputationComputationStrategy* and *StrategyAverageRatings* respectively, can be seen in listing 6.8. The interface defines one function, *compute*, that all ConcreteStrategies must implement in order to be used as a computation strategy. *compute* takes a number of parameters, all of which are a resource's raw user ratings in each of the categories users can rate a resource on. Thus, the job of a ConcreteStrategy's compute function is to calculate a reputation score for each of the rating types. In this prototype we chose to implement the simple average of ratings algorithm, which was discussed in Chapter 2 Section 2.4.3. As we have already discussed, another algorithm can easily replace it by creating a new class that implements the *IReputationComputationStrategy* interface and setting the *_currentStrategy* variable in the *ComputationEngine* class to the instance of the new ConcreteStrategy. Any call on *computeResourceReputation* from listing 6.5 will now use the new algorithm to compute a resource's reputation scores.

The implementation details of the average of ratings algorithm can be seen in the *compute* function in listing 6.8. This function also has a few auxiliary functions; *computeQualityScore*, *computeImportanceScore*, *computeRelevanceScore* and *makeDefaultRelevanceScoreArray*. We have chosen not to show them all in their entirety due to redundancy as they are quite similar. Notice that in the *compute* function if there exists a Michelin rating, then both the resource's quality score and importance score is set to 10 and the other raw ratings do not matter. However, Michelin ratings do not influence the relevance score of a resource. This is because

resources that have received a Michelin rating should be regarded as extraordinary and should always be ranked above resources that have not received Michelin ratings.

If a resource has not received any quality ratings, then the quality score is set to 1, which is the lowest possible score. The same is true for importance ratings. The last outcome is if a resource has quality ratings, but no Michelin ratings. Then, the auxiliary function *computeQualityScore* will compute the average of all quality ratings and set the result as the resource's quality score.

Calculating the relevance score of a resource differs from calculating quality and importance because relevance ratings exist in two forms. One from ratings that the user gives a resource and one that is given to the resource at the time it was added to the system. In addition, a resource has a relevance for each topic in the system. Even if no user has submitted a relevance rating, the resource's default relevance score for each topic is still computed.

Listing 6.8: The StrategyAverageRatings class

```
1  package interfaces {
2      public interface IReputationComputationStrategy {
3          function compute(qualityRatings:*, importanceRatings:*, relevanceRatings:*,
               michelinRatings:*, defaultRelevanceRatings:*):Array;
4      }
5  }
6
7  package {
8      import interfaces.IReputationComputationStrategy;
9
10     public class StrategyAverageRatings implements IReputationComputationStrategy {
11
12         public function StrategyAverageRatings() {}
13
14         public function compute(qualityRatings:*, importanceRatings:*,
               relevanceRatings:*, michelinRatings:*, defaultRelevanceRatings:*):Array
                {
15             var scoreArray = new Array();
16
17             if (michelinRatings) scoreArray["qualityScore"] = 10;
18             else if (!qualityRatings) scoreArray["qualityScore"] = 1;
19             else scoreArray["qualityScore"] = computeQualityScore(qualityRatings);
20
21             if (michelinRatings)scoreArray["importanceScore"] = 10;
22             else if (!importanceRatings) scoreArray["importanceScore"] = 1;
23             else scoreArray["importanceScore"] = computeImportanceScore(
                   importanceRatings);
24
25             if (!relevanceRatings) {
26                 scoreArray["relevanceScore"] = makeDefaultRelevanceScoreArray();
27                 for (var i:int = 0; i < scoreArray["relevanceScore"].length; i++) {
28                     for (var j:int = 0; j < defaultRelevanceRatings.length; j++) {
29                         if (scoreArray["relevanceScore"][i]["topicID"] ==
                               defaultRelevanceRatings[j]["topicID"]) {
30                             scoreArray["relevanceScore"][i]["score"] =
                                   defaultRelevanceRatings[j]["defaultRelevance"];
31                         }
32                     }
33                 }
34             }else scoreArray["relevanceScore"] = computeRelevanceScore(
                   relevanceRatings, defaultRelevanceRatings);
35
36             return scoreArray;
37         }
38
39         private function computeQualityScore(qualityRatings:*):Number {
```

```
40              var sum = 0;
41              for (var i:int = 0; i < qualityRatings.length; i++) {
42                  sum += qualityRatings[i]["rating"];
43              }
44
45              return sum / qualityRatings.length;
46          }
47
48          private function computeImportanceScore(importanceRatings:*):Number {
49              /* Implementation details similar to computeQualityScore */
50          }
51
52          private function computeRelevanceScore(relevanceRatings:*,
                defaultRelevanceRatings:*):Array {
53              /* Similar to computeQualityScore except that we go through two arrays
                    instead of one */
54          }
55
56          private function makeDefaultRelevanceScoreArray():Array {
57              var relevanceScores:Array = new Array();
58              var topicIDs:Array = DomainMapObject.getInstance().topicIDs;
59
60              for (var i:int = 0; i < topicIDs.length; i++) {
61                  relevanceScores[i] = new Array();
62                  relevanceScores[i]["topicID"] = topicIDs[i];
63                  relevanceScores[i]["score"] = 0;
64              }
65              return relevanceScores;
66          }
67      }
68  }
```

The final part of the reputation engine is the QueryEngine class. This class is responsible for finding the appropriate resources for a query and returning the resulting set so that they can be displayed to the user who made the query. Naturally, queries are made either through the manipulation of sliders, keyword searches, or the combination of both. Queries can be made over both knowledge resources and user resources. Since the sliders for the two resource types are different, the queries are necessarily different as well. Thus, we must separate the query implementation into two different functions. One for knowledge resources and one for user resources.

We show the show the implementation of the knowledge resource query functions in listing 6.9. *computeQueryResult* is the main function, and uses the auxiliary functions *checkIfResourceSatisfiesThresholds*, *checkIfResourceSatisfiesSearchTerm*, and *computeQRI*. Obviously, we are interested in finding the knowledge resources that satisfies the thresholds a user has set using the available sliders as well as the search term from an eventual keyword search. All knowledge resources who satisfy these conditions have their QRI-scores computed and added to a query result set. This set is then sorted on the knowledge resource's QRI-score in descending order so that the resources with the highest scores appear first. Lastly, the result set is return to the DomainMapObject, which called the *computeQueryResult* function, and the resources contained in the result set are displayed on the screen.

The implementation details of the user resource query functions are not shown here because they are very similar to the functions used by the knowledge resource query implementation. Main differences include the fact that we filter user resources instead of knowledge resources, and that the sliders users use to set thresholds for user resources differ from those for knowledge resources. Naturally, the rest of the source code for the reputation engine classes can be found in Appendix A.

Listing 6.9: Knowledge resource query functions

```
1  public function computeQueryResult(resourceCollection:ResourceCollection):
       ResourceCollection {
2    var queryResult:ResourceCollection = new ResourceCollection();
3    var iter:IIterator = resourceCollection.iterator();
4
5    while (iter.hasNext()) {
6      var nextResource:* = iter.next();
7      var satisfiesThreshold:Boolean = checkIfResourceSatisfiesThresholds(
           nextResource);
8      var satisfiesSearchTerm:Boolean = checkIfResourceSatisfiesSearchTerm(
           nextResource);
9
10     if (satisfiesThreshold && satisfiesSearchTerm) {
11       var qri = computeQRI(nextResource);
12       if(qri > 0){
13         nextResource.qriScore = qri;
14         queryResult.addElement(nextResource);
15       }
16     }
17   }
18
19   queryResult.sort();
20   return queryResult;
21 }
22
23 private function checkIfResourceSatisfiesSearchTerm(resource:*):Boolean {
24   if (_searchTerm == "") return true;
25   else {
26     var keywords:Array = resource.keywords;
27
28     for (var i:int = 0; i < keywords.length; i++) {
29       var word = keywords[i]["keyword"];
30       if (word.toLowerCase() == _searchTerm.toLowerCase()) return true;
31     }
32     return false;
33   }
34 }
35
36 private function checkIfResourceSatisfiesThresholds(resource:*):Boolean {
37   if (resource.qualityScore >= _userQualityTreshold && resource.importanceScore >=
         _userImportanceTreshold && resource.expertise >= _userKnowledgeTreshold) {
38     var resourceUpdateTime = resource.updateTime;
39     var now = new Date().time / 1000;
40
41     if (_userTimeTreshold == 1) return true;
42     else if(_userTimeTreshold == 2 && resourceUpdateTime >= (now - (86400 * 180)))
           return true;
43     else if(_userTimeTreshold == 3 && resourceUpdateTime >= (now - (86400 * 30)))
           return true;
44     else if(_userTimeTreshold == 4 && resourceUpdateTime >= (now - (86400 * 7)))
           return true;
45     else return false;
46   }else return false;
47 }
48
49 private function computeQRI(resource:*):Number {
50   var qri = resource.qualityScore * resource.relevanceScore * resource.
         importanceScore;
51   return qri;
52 }
```

SystemMessages

The *SystemMessages* class is responsible for displaying messages to users when applicable. Listing 6.10 shows the source code of the most important functions. The *displayMessage* function is public, making at available for other classes. *displayMessage* takes a *String* parameter, which is the message that is displayed to users and makes it visible. Notice that *this.messageText* is a text field element created in the Flash environment and added to the SystemMessages class. It is referenced to using the *this* keyword. *text* is a property of the text field and takes the string that is displayed.

Two other important functions in this class are *show* and *hide*, which makes the entire SystemMessages window visible and invisible, respectively. *show* is invoked when another class in the prototype calls the *displayMessage* function, while *hide* is called when a user clicks on an exit button that is present in the window.

Listing 6.10: SystemMessages class functions

```
1   public function displayMessage(message:String):void {
2     this.messageText.text = message;
3     show();
4   }
5
6   private function show():void {
7     this.visible = true;
8   }
9
10  private function hide():void {
11    this.visible = false;
12  }
```

6.3.3 The Collections

In the previous chapter we discussed how all Collection classes uses the Iterator pattern in order to traverse its elements. The Iterator pattern consists, as we remember, of the *Iterator* and the *Aggregate* as well as the concrete implementation of these. In listing 6.11 we see the *IIterator* interface, while listing 6.12 shows as a concrete implementation called *ArrayIterator*. Naturally, it is responsible for iterating through the elements of an array, which it receives through its constructor, in a bottom-up fashion. Due to the properties of the interface concept and the Iterator pattern, we could make additional concrete iterator implementations that traverses the elements in other ways than *ArrayIterator* does. However, for our current need we only require a simple bottom-up iterator.

Listing 6.11: Iterator interface

```
1 public interface IIterator {
2   function reset():void;
3   function next():Object;
4   function hasNext():Boolean;
5   function setIndex(value:int):void;
6 }
```

The functions that a concrete implementation of IIterator must implement are quite simple. *Reset* naturally resets the iterator to its default state. *Next* fetches the next element and updates the iterator state to get the following element the next time the function is called. *hasNext* simply checks if there are more elements to fetch. If there is, the function returns true, while it returns

false if every element has been iterated through. Finally, *setIndex* sets the iterator state to the integer given by the value parameter. For example, if 10 is the parameter, then the next element that is fetched will be element in place number 10 in the array.

Listing 6.12: Iterator implementation

```
1 public class ArrayIterator implements IIterator {
2    private var _index:uint = 0;
3    private var _collection:Array;
4
5    public function ArrayIterator(collection:Array) {
6      _collection = collection;
7      _index = 0;
8    }
9
10   public function hasNext():Boolean {
11     return _index < _collection.length;
12   }
13
14   public function next():Object {
15     return _collection[_index++];
16   }
17
18   public function reset():void {
19     _index = 0;
20   }
21
22   public function setIndex(value:int):void {
23     _index = value;
24   }
25 }
```

The other part of the Iterator pattern is the *Aggregate*. Listing 6.13 shows the *ICollection* interface, which is our Aggregate, while listing 6.14 shows the ConcreteAggregate *ResourceCollection*. Other ConcreteAggregate's in our system includes *TopicCollection*, *CommentCollection* and *UserCollection*. All of these have a roughly similar implementation, with some having additional functions. The *ICollection* interface is very simple. It has one function that all ConcreteAggregates must implement, namely *iterator*. This function creates and returns the appropriate iterator given by the *type* parameter.

ResourceCollection's implementation contains two important functions. First, we have the *iterator* function from the *ICollection* interface. Notice that since ArrayIterator is our only ConcreteIterator we do not care about the function parameter. We just make an instance of ArrayIterator with the *_data* variable as the elements to iterate over and return it. The other function common to all collection implementations is *addElement* which simply takes an object and adds it at the end of the _data array. In this case, the object is a *DomainResource* object, but the object type will vary depending on the collection. *sort*, *size* and *reset* are functions that not all collections have. The two latter functions should be self-explanatory. *size* returns the number of elements in the collection, while *reset* empties the collection. The last function, *sort*, sorts the collection in descending order on the property *qriScore*, which is a variable that each DomainResource object has.

Listing 6.13: Aggregate interface

```
1 public interface ICollection {
2   function iterator(type:String = null):IIterator;
3 }
```

Listing 6.14: Aggregate implementation

```
1  public class ResourceCollection implements ICollection {
2    private var _data:Array;
3
4    public function ResourceCollection() {
5      _data = new Array();
6    }
7
8    public function addElement(resource:DomainResource):void {
9      _data.push(resource);
10   }
11
12   public function iterator(type:String = null):IIterator {
13     return new ArrayIterator(_data);
14   }
15
16   public function sort():void {
17     _data.sortOn("qriScore", Array.NUMERIC | Array.DESCENDING);
18   }
19
20   public function size():int {
21     return _data.length;
22   }
23
24   public function reset():void {
25     _data.splice(0, _data.length);
26   }
27 }
```

The elements that are contained within the various collections are *DomainResource*, *DomainTopicObject*, *ResourceComment* and *User* objects. These objects contain various data and are linked to a graphical element that is drawn onto the client screen. For example, a DomainResource object contains data such as its name, an URL that points to the resource's physical location and its current QRI-score among other things. Each DomainResource also contains a ResourceValueMatrixObject instance. This object contains the raw data that is related to the value of a DomainResource such as ratings, when it was last updated and resource comments. Other parts of the prototype uses this data to display the correct DomainResouces given a query. Furthermore, DomainResources contain code that enables user interaction. For example, clicking on a DomainResource will update the right side panel with information related to the DomainResource as well as showing ResourceComment objects on the screen. For the implementation details of the collection elements, please see the source code in Appendix A. We have chosen not to show it here because the implementation is mainly concerned about assigning values to variables and making the variables available through set and get functions.

6.3.4 Domain Map Object

Finally in this section we will talk about DomainMapObject class. As we have mentioned before, this is the class that holds everything together. Most communication between different parts of the prototype goes through this class. This is easily done because of the fact that it is a singleton, thus other classes can invoke functions in the DomainMapObject class by saying *DomainMapObject.getInstance().functionName()*. For example, a click on a DomainResource needs to trigger a function in the GUI part that updates the right side panel with the correct data. Since the DomainResource class and the GUIViewStateMachine class are located in separate parts of the prototype, this communication needs to pass through the DomainMapObject, which has access to all parts of the system since it instantiates all the other major classes.

The DomainMapObject is also responsible for drawing most graphical elements to the screen. After the prototype has loaded data from the database and initialized appropriate classes, the DomainMapObject iterates through the TopicCollection and draws all its elements to the screen. These elements are now interactable and once clicked, the DomainMapObject will iterate through the ResourceCollection and draw its elements in place for the TopicCollection elements. Finally, exactly the same is done with the Commentcollection when a ResourceCollection element is clicked. Listing 6.15 shows the algorithm that draws DomainResources on the screen when the function *drawFocusTopicAndResources* is called. This function is invoked when either a DomainTopic is clicked, or a user performs a new query by manipulating the control panel sliders.

For more details about the implementation of the DomainMapObject class, or other classes that we have talked about in this section, please see Appendix A.

Listing 6.15: The drawFocusTopicAndResources function

```
1  public function drawFocusTopicAndResources(focusTopicID:int):void {
2    removeDrawnObjects(); //Remove the objects that are currently drawn on the screen
         .
3
4    var iter:IIterator = _topicCollection.iterator();
5
6    //we find the topic object given by the focusTopicID variable
7    while (iter.hasNext()) {
8      var nextTopic:* = iter.next();
9      if (nextTopic.id == focusTopicID) break;
10   }
11
12   _activeTopicID = focusTopicID;
13   nextTopic.x = 435; //we correctly place the topic on the screen
14   nextTopic.y = 227;
15
16   _drawnObjects.push(nextTopic);
17   addChild(nextTopic); //adds the topic to the screen
18
19   /* Find the knowledge resources that matches the current query */
20   _queryResourcesCollection = _queryEngine.computeQueryResult(_resourceCollection);
21   iter = _queryResourcesCollection.iterator();
22   iter.setIndex(_resourceCounter);
23
24   /* For as long as there is space on the screen, draw resources on it */
25   while (iter.hasNext() && (_resourceCounter < _resourceCap)) {
26     var nextResource:* = iter.next();
27     drawResource(nextResource, nextTopic);
28     _resourceCounter += 1;
29   }
30 }
31
32 private function drawResource(resource:*, topic:*):void {
33   addChild(resource); //draw to the screen
34
35   var radius:Number = ((topic.hitbox.width + resource.width) / 2) + 40;
36   var angle:Number = (_drawnObjects.length / 10) * 2 * Math.PI;
37
38   /* place the resource in a circulater pattern around the topic */
39   resource.x = topic.x + Math.round(radius * Math.sin(angle));
40   resource.y = topic.y + Math.round(radius * Math.cos(angle));
41
42   _drawnObjects.push(resource);
43 }
```

Users table	
Column	Description
userID	An id that is automatically given to a user when he registers. Is used to uniquely identify a user.
email	A user's email that is provided upon account registration. Is used as a part of the user's login credentials and as a way to contact the user.
password	The password that a user provides upon registration, saved in MD5 encryption. Is used as a part of the user's login credentials.
firstName	A user's first name.
lastName	A user's last name.
country	A user's country of residence.
employer	The place where a user has stated that he works.
position	A user's work title or position.
role	For future use. Determines a user's role in the system. Default value is "user", but can also be "expert" or "admin".
contributionPoints	A number which says something about how much a user has contributed to the system. All positive actions earn points. Plays a part in a user's value matrix.

Table 6.1: The details of the Users database table

6.4 The Server

In this section, we will talk about the server and the database in more detail. First, we will discuss the details of the various tables in the database, before moving on to present the other files that make up the server side functionality of the prototype.

6.4.1 The Database

In Chapter 4, we briefly discussed why a rational SQL database was chosen to save our data. Also, we presented the database structure in the form of an UML diagram. We will in this subsection go into further details of the various tables, discussing their overall purpose as well as explaining their attributes. Currently, the database is located on servers owned by University of Oslo. For details that are not provided here, please see Appendix B which contains information about how to access the database through the phpMyAdmin GUI.

The first table we will look at is the *Users* table. It's main role is to store all users that have registered with the prototype, as well as enable users to login when they provide their credentials upon request. Furthermore, the Users table keep track of a user's contribution score, which is updated whenever the user interacts with the domain map by adding new resources, evaluating resources and so on. Table 6.1 provides more details about the columns in the Users table.

Next, we have the *DomainTopic* table. This table simply stores the topics of a domain map. Details about its attributes can be seen in table 6.2.

Continuing, we present the *Resource* table. Naturally, this table keeps track of all resources that have been added by users during the prototype's lifetime. Table 6.3 presents the details of Resource's columns. Other than being created when a resource is added, a row also updates its

DomainTopic table	
Column	Description
ID	The id that is automatically given to a DomainTopic when it is added. Is used to uniquely identify a DomainTopic.
TopicName	A DomainTopic's name. Its purpose is to be human readable so users can identify a DomainTopic by its name.
mapID	Currently not in use, but the idea is that many domain maps can share the same database and tables and each map is identified by this id.

Table 6.2: The details of the DomainTopic database table

"updateDate" attribute whenever a user interacts with the resource that the rows represents in the prototype.

Following, we have two tables that store data that is a part of a resource's value matrix, namely table table 6.4 and table 6.5. The former saves data that is related to comments that users provide on resources. Other than the data necessary to restore the comment object in the client, this table also keeps track of how many "likes" a comment has received. Thus, a row in this table is updated whenever a user interacts with then "like" functionality on comments in the client. The latter table keeps track of keywords that users have provided for a resource. Other than an ID to identify which resource the keywords belongs to as well as the keyword itself, the table also saves how many times a particular keyword has been submitted. We do this so we can sort the result set from a keyword query on how many times the keyword has been submitted. In other words, we infer that resources which have had the keyword "reputation" submitted five times is more relevant to a query than resources that have had the same keyword submitted less than five times.

Table 6.6 is an interesting one. We found that this table is required because a resource needs a default relevance score for at least one topic in order for the resource to show up in the client at all since at the beginning of a resource's lifetime no ratings have been provided yet. A new row is added to the table under two different circumstances. The first is when a new resource is added, then the user who adds the resource is required to provide which topic the resource belongs to. Otherwise, a row is added when a user, through the client GUI, associates a resource with a previously unassociated topic. Currently, the value of the defaultRelevance attribute counts as the same as a single rating, but this can change by giving the defaultRelevance value a higher weight when computing the final relevance score for a resource. In addition to the defaultRelevance attribute, Table 6.6 also contains two IDs, resourceID and topicID, that are used to identify the resource and in which topic the defaultRelevance applies to.

The last four tables, detailed in table 6.7, table 6.8, table 6.9 and table 6.10, are all very similar in both structure and purpose. Their function is to store data related to the rating of resources, and are, therefore, a part of each resource's value matrix. All of the tables need to know who has provided the rating and for which resource it applies to, in addition to the rating itself. We notice that table 6.7 differs slightly in structure from the others as it has an additional attribute in "topicID". The reason for this is obviously that a resource can belong to several topics and we need to know to which topic the rating applies. We do not need this information in, for example, the table that stores quality ratings because the quality of a resource can be rated regardless of the topic it belongs to.

Resource table	
Column	Description
id	The id that is automatically given to a Resource when it is added. Is used to uniquely identify a Resource.
resourceName	A Resource's name. Its purpose is to be human readable so users can identify a Resource by its name.
url	A pointer to the physical location of a resource. Through it, users can read the resource's content.
authoreName	The name of the resource's author. Not neccessarily the same person that added it.
addedBy	Foreign key to the Users table. Is used to identify that user that has added the Resource.
creationDate	The date the resource was added to the database.
updateDate	The date that the resource was last interacted with.
expertiseLevel	The expertise level that was provided by the user who added the resource. Enables users to make a distinction between resources on how difficult they are to understand.

Table 6.3: The details of the Resource database table

ResourceComments table	
Column	Description
id	The id that is automatically given to a ResourceComment when it is added. Is used to uniquely identify a ResourceComment.
resourceID	Foreign key to the Resource table. Is used to identify the resource to which the comment has been provided.
userID	Foreign key to the Users table. Is used to identify the user who provided the rating.
title	A ResourceComment's title.
text	A ResourceComment's content, i.e. the comment itself.
likes	The amount of likes a ResourceComment has received. Is used to rank ResourceComments in descending order.

Table 6.4: The details of the ResourceComments database table

ResourceKeywords table	
Column	Description
resourceID	Foreign key to the Resource table. Is used to identify the resource to which the rating is provided.
keyword	The keyword that has been provided by a user.
amount	The amount of times the keyword has been provided. Is used to rank resources that have the same keywords.

Table 6.5: The details of the ResourceKeywords database table

DefaultResourceRelevance table	
Column	Description
resourceID	Foreign key to the Resource table. Is used to identify the resource to which the default relevance applies.
topicID	Foreign key to the DomainTopic table. Is used to identify which topic the default relevance applies to.
defaultRelevance	This number is added in additional to other relevance values. Currently, the only possible value is 1, meaning that the resource resourceID is relevant in the topic topicID.

Table 6.6: The details of the DefaultResourceRelevance database table

ResourceUserRelevanceRating table	
Column	Description
userID	Foreign key to the Users table. Is used to identify the user who provided the rating.
resourceID	Foreign key to the Resource table. Is used to identify the resource to which the rating is provided.
topicID	Foreign key to the DomainTopic table. Is used to identify in which topic the rating was provided.
rating	The rating itself. The value of the rating is either 0 or 1. 0 if the user who provided the rating thinks the resource is not relevant to the topic, or 1 if he thinks the resource *is* relevant.

Table 6.7: The details of the ResourceUserRelevanceRating database table

UserImportanceRatings table	
Column	Description
userID	Foreign key to the Users table. Is used to identify the user who provided the rating.
resourceID	Foreign key to the Resource table. Is used to identify the resource to which the rating is provided.
rating	The rating itself. For importance it is a number between 1 and 10. 1 means low importance, while 10 means extremely important.

Table 6.8: The details of the UserImportanceRatings database table

UserMichelingRatings table	
Column	Description
userID	Foreign key to the Users table. Is used to identify the user who provided the rating.
resourceID	Foreign key to the Resource table. Is used to identify the resource to which the rating is provided.
rating	The rating itself. The value is only ever 1, because a user has either given a resource a michelin star or he has not. There is no reason to store instances where a rating has not been provided.

Table 6.9: The details of the UserMichelingRatings database table

UserQualityRatings table	
Column	Description
userID	Foreign key to the Users table. Is used to identify the user who provided the rating.
resourceID	Foreign key to the Resource table. Is used to identify the resource to which the rating is provided.
rating	The rating itself. For quality it is a number between 1 and 10. 1 means low quality, while 10 means exceptional quality.

Table 6.10: The details of the UserQualityRatings database table

6.4.2 Server Files

In addition to the database, the server contains a few other files. Most important is *software.swf*, where the .swf extension denotes that this is a compiled Adobe Flash file that contains source code as well as graphical elements. In other words, *software.swf* is the prototype application itself. Then, we have *index.php* which serves two purposes. First, it embeds software.swf so that the prototype application is automatically started whenever the server's index, or main, page is visited. Naturally, the index page is often the first page that a user sees when entering a web site. Currently, our index page, and therefore the location of the prototype, is located at *http://benjamij.at.ifi.uio.no/*.

The other purpose is to enable links clicked on in the prototype application to be opened in a new tab in a user's browser. Since this is something that needs to be done client side, we need to have a piece of Javascript code that opens a new browser tab and points the tab to the URL provided by the prototype application. Luckily, PHP-scripts can contain HTML which again can contain Javascript. Listing 6.16 shows a snippet of code from *index.php* that fulfills the wanted functionality.

Listing 6.16: Index.php - Javascript that opens a new browser tab

```
<script type="text/javascript">
  function goToUrl(url, name){
    window.open(url, name);
  }
</script>
```

We have already talked about the other software that resides on the server side of the application, AmfPHP and PHPMyAdmin, earlier in this chapter. However, we need to talk about *VMOServer.php*, which is the script that AmfPHP uses to communicate between the database and the client application. It is located inside the *Services* folder in the AmfPHP directory. The shell of such a service script is shown in listing 6.17.

Listing 6.17: The shell of an AmfPHP service script

```
<?php

class VMOServer {
/* Place class variables here */

  public function __construct(){
    /* Set the class variables.
     *  Prepare the class for use.
     *  For example create a connection to a database.
     */
  }

  /* Place functions here that you want the client
   * to access through AmfPHP
   */
}
?>
```

Naturally, the script needs to have a set of functions that the client can call upon to either insert data to the database or extract data from it. If needed, the script returns some data back to the client. Listing 6.18 and listing 6.19 each show an example of such a function. The former is responsible for fetching data related to the resources and returning it to the client so that resource objects can be generated and displayed to the users, while the latter receives some

resource data from the client and saves it to the database. If no errors occur, then the result from the call on saveResource() from listing 6.19 is that a new resource is created and will be fetched on the next call on getResources() from listing 6.18. The return value from saveResource() is either *true* or *false*, and a message will be displayed to the end user saying that a new resource has been successfully added to the database or that an error has occurred, depending on the return value.

Listing 6.18: The function getResources() in VMOServer.php

```
1  public function getResources(){
2
3    $retArray = array();
4    $sql = "SELECT * FROM Resource";
5    $result = mysql_query($sql);
6
7    $resourceCounter = 0;
8    while($row = mysql_fetch_array($result, MYSQL_ASSOC)){
9      $resourceID = $row["id"];
10
11     $retArray[$resourceCounter]["resourceData"]["id"] = $resourceID;
12     $retArray[$resourceCounter]["resourceData"]["resourceName"] = $row["
           resourceName"];
13     $retArray[$resourceCounter]["resourceData"]["url"] = $row["url"];
14     $retArray[$resourceCounter]["resourceData"]["author"] = $row["authorName"];
15     $retArray[$resourceCounter]["resourceData"]["expertiseLevel"] = $row["
           expertiseLevel"];
16     $retArray[$resourceCounter]["resourceData"]["updateDate"] = $row["updateDate"];
17
18     $sql = "SELECT keyword FROM ResourceKeywords WHERE resourceID = '$resourceID'
           ORDER BY amount DESC LIMIT 5";
19     $result2 = mysql_query($sql);
20
21     while($row2 = mysql_fetch_array($result2, MYSQL_ASSOC)){
22       $retArray[$resourceCounter]["keywords"][] = $row2;
23     }
24
25     $sql = "SELECT u.firstName, u.lastName, rc.title, rc.text, rc.likes, rc.id FROM
            ResourceComments rc, Users u WHERE resourceID = '$resourceID' AND u.userID
            = rc.userID";
26     $result2 = mysql_query($sql);
27
28     while($row2 = mysql_fetch_array($result2, MYSQL_ASSOC)){
29       $retArray[$resourceCounter]["resourceComments"][] = $row2;
30     }
31
32     $sql = "SELECT topicID, defaultRelevance FROM DefaultResourceRelevance WHERE
           resourceID = '$resourceID'";
33     $result2 = mysql_query($sql);
34
35     while($row2 = mysql_fetch_array($result2, MYSQL_ASSOC)){
36       $retArray[$resourceCounter]["defaultRelevanceRatings"][] = $row2;
37     }
38
39     /*The rest of the queries are very similar to the one above except that they
           extract other ratings.*/
40
41     $resourceCounter += 1;
42   }
43
44   return $retArray;
45 }
```

Listing 6.19: The function saveResources() in VMOServer.php

```
1  public function saveResource($resourceData){
2      $retArray = array();
3
4      $now = time();
5      $resourceName = $resourceData["name"];
6      $resourceURL = $resourceData["url"];
7      $expertiseLevel = $resourceData["expertise"];
8      $userID = $resourceData["userID"];
9      $name = $resourceData["topic"];
10     $author = $resourceData["author"];
11
12     $sql = "INSERT INTO Resource (resourceName, url, authorName, addedBy,
           creationDate, updateDate, expertiseLevel) VALUES ('$resourceName', '
           $resourceURL', '$author', '$userID', '$now', '$now', '$expertiseLevel')";
13     $result = mysql_query($sql);
14
15     $resourceID = mysql_insert_id();
16
17     if(mysql_affected_rows() > 0){
18         $retArray["success"] = true;
19     }else $retArray["success"] = false;
20
21     $sql = "SELECT ID FROM DomainTopic WHERE TopicName = '$name'";
22     $result = mysql_query($sql);
23     $row = mysql_fetch_array($result, MYSQL_ASSOC);
24
25     $topicID = $row["ID"];
26
27     $sql = "INSERT INTO DefaultResourceRelevance (resourceID, topicID) VALUES ('
           $resourceID', '$topicID')";
28     $result = mysql_query($sql);
29
30     return $retArray;
31 }
```

Chapter 7

Prototype Evaluation

"I didn't fail the test, I just found 100 ways to do it wrong. "

– Benjamin Franklin

In this chapter, we start out with talking about common ways of testing and evaluation software, before presenting our evaluation method. Then, we present our evaluation set up and ground work, before we finally present a summary of our results.

7.1 Evaluation Method

In computer science, there are many ways to evaluate software and many areas to evaluate. We often associate software testing and evaluation with the activity of uncovering bugs and errors in the source code, or finding if the software fulfills a set of requirements [54][2]. These activities are often included in the term software verification [105]. On the other hand, we have software validation techniques, which are more concerned about if the software and its requirements satisfy the needs of the end users [105].

Naturally, since we have created a prototype only, we are not interested in uncovering bugs and such. Rather, we are interested in what others think of our prototype, and at its core the Value Matrix Object and Domain Map Object concepts. More specifically, we want to investigate the boundary object properties of our two objects. We remember from Chapter 1 that a boundary object is an object that acts as a communication channel between two communities.

For our purpose, we cannot use traditional software evaluation techniques. The closest technique that could have been applicable is usability testing. This technique involves observing and recording users interacting with a product in a controlled environment as well as asking them questions, either in an interview or using questionnaires, related to the product they test [92]. However, setting up such a test environment is difficult for us since we need to involve people from different communities who live in different parts of the world. In addition, usability testing involves end users, while we want to involve members of communities that will help further develop the Value Matrix Object and the Domain Map Object to create a new socio-technical system. Because of this, our evaluation method is closer to that of qualitative research than usability testing, where we use questionnaires as our data collection method [24].

7.2 Evaluation Setup

As mentioned, we are interested in testing our presented objects' role as boundary objects. That is, to enable communication between two technical communities. For this to be possible, we need to let actual people from such communities try our prototype. For our purpose, we define two communities that we invite to help us evaluate the prototype. These are "tool makers" and "systemic innovators". The systemic innovators belong to a multidisciplinary group of people who work to create innovative real-life socio-technical systemic solutions for education, academic research and so on. Tool makers, on the other hand, are experts that work with technology such as Topic Maps and Semantic Web.

In order to enable people from the two communities to have access to the prototype, we needed to find an environment where it could be gained access to regardless of the physical location of the evaluators. Obviously, the Internet is a perfect environment for this. Thus, we created a web page and integrated the prototype into it. Then, we created a couple of dummy profiles that our evaluators could use to log in and explore the prototype with.

Naturally, after the evaluators had tried the prototype, we needed some way to gather their opinions. For this purpose we decided to create two different questionnaires that we asked the evaluators to answer, one for the systemic innovators and one for the tool makers. Systemic innovators were asked to answer questions related to the underlying functionality of the prototype, rather than design and implementation questions which were reserved for the tool makers. The questionnaires were published on the Internet using the SurveyMonkey software. By doing this, we made it easy for the evaluators to answer the questions and an easy for us to collect their answers. The questionnaire for the systemic innovators is shown in table 7.1, while the questionnaire meant for the tool makers can be seen in table 7.2. In addition to the questions listed, we also provided an open text field where we asked the evaluators to make any comment or suggestion they thought was not covered in the questions.

Finally, we provided access to both the questionnaires and the prototype through an email addressed to all the participating evaluators. Email was a suitable platform for this because not only did we have to provide links to the prototype and questionnaires, but also some information about the project. Of course, the evaluators needed some context and understanding about the Value Matrix Object and the Domain Map Object as well, in order to be able to properly explore the prototype and answer the questionnaires. Additionally, email provides a direct link between us and each evaluator so that potential questions could be easily answered. The content of the email each evaluator was sent can be found in Appendix C.

7.3 Results

After the evaluators had been given sufficient time to answer the questionnaires, we collected their answers and analyzed them. Unfortunately, we only received answers from four evaluators, three from tool makers and one from systemic innovators. Still, the feedback was sufficient to give a rough overview over the areas of the prototype that needs to improve.

The most alarming feedback was given by the tool makers. All three of the responders thought that the prototype was not intuitive enough and felt confused. They simply did not know what to do when they first tried it. One evaluator suggested that he would be less confused if functionality, such as the sliders, is only shown when it can actually be used. Currently, there are some cases where buttons and sliders are shown, but do not do anything in the current context. Another user called for documentation of how the prototype was meant to be used. Clearly, both of these measures should be considered to improve the prototype. For example, providing an example of usage would give users time to get familiar with the prototype and gain

Questions for systemic innovators	
Number	Question
1	Currently, there is a set of sliders that modify what resources are shown on the screen. Can you think of any other affordances that could do the job better?
2	Can you think of any relevant criteria other than time, knowledge, quality and importance that should have its own slider?
3	At the moment, users are rewarded within two categories. Either by contributing to the organization of existing knowledge by evaluation or by publishing new material. Can you think of other ways to reward users so that they feel inclined to use this prototype or a similar tool?
4	In the prototype, a knowledge resource's value in a context is computed using quality, importance and relevance dimensions. These dimensions are given by users in the form of ratings which a reputation engine collects and computes a score for. Can you think of other dimensions that could be relevant to the computation of a knowledge resource's value?
5	Can you think of other ways to evaluate the quality, importance and relevance dimensions other than through user ratings?
6	As a systemic innovator, is the functionality of the prototype sufficient in order to build systems that create an ecology of knowledge work? If not, what do you think is missing?

Table 7.1: Questions we asked systemic innovators to answer

Questions for tool makers	
Number	Question
1	When you first tried the prototype was it clear what you were supposed to do?
2	While using the prototype, did you feel overwhelmed by the amount of information provided on the screen? If yes, what would you change?
3	Currently, users are met with a set of topics they can click on in order to see the knowledge resources that are relevant to that topic. Further, they can click on a resource to view more information about it and evaluate it. Is this layout intuitive and understandable, or should it change? If yes, do you have any suggestions on what should be improved?
4	Are the sliders that modify what resources are shown understandable and intuitive? What can be done to improve how users filter the resources?
5	From the viewpoint of a toolmaker, does the prototype provide all the functionality needed for the intended usage? If not, what is lacking?
6	From the viewpoint of a toolmaker, if you could change anything about the prototype, what would it be?

Table 7.2: The questions we asked tool makers to answer

a better understanding of it so they can more easily explore it on their own afterwards. Further, the tool maker evaluators for the most part liked the sliders and thought the concept intuitive, but wanted an explanation of what each of them actually did. The conclusion we can draw from the tool maker evaluators is that this prototype is a good start, but the GUI must improve to make it more intuitive and less ambiguous.

As mentioned, only one person answered the systemic innovator questionnaire. Still, this person raised some good points and suggestions. His biggest concern was that user rewards were not clear enough in the prototype. That is, the user is never explicitly shown how he is rewarded for his actions. He further suggests that it is possible to use mechanics from computer games for rewarding users. Furthermore, the evaluator suggests that other sliders such as ”popularity“ or ”number of connections“ can be introduced in addition to those that already exists for knowledge resources. Finally, he provides his encouragements, saying that the prototype is an important first step, and the first operational prototype that attempts to tackle the complex issue of acquiring information necessary to assess the value proposition for resources in an epistemic ecosystem. The biggest challenge for the future, he says, is to make the tool simple enough for the average user to participate. The more tasks we ask users to perform, the lower the probability that they will perform them.

All in all, this evaluation produced a set of constructive criticism that was clearly needed so improvements can be made in the future. An exhaustive presentation of the questions and their answers can be found in Appendix C.

Chapter 8

Discussion and Further Work

> "No, I don't want you to draw any conclusion. I want you to listen to what I just said."
>
> – Joe Morgan

In this last chapter we provide a summary of this thesis and our contribution. Then, we look back on the entire process from beginning to end, discussing what we have learned and what could have been done differently. Finally, we look at how our work may be continued, both in terms of theoretical work and enhancement of our prototype tool.

8.1 Contribution

The purpose of this thesis has been to present two objects, the Value Matrix Object and the Domain Map Object, which we propose to use as building blocks in new sociotechnical systems tailored to help us solve information overload. Naturally, the two objects can help solve other global problems as well since the act of organizing information is key to solving any problem. In short, the Value Matrix Object collects all data that is used to compute the value of an information resource, while the Domain Map Object is used to organize information resources.

We have pointed out that already existing technology is lacking when put into use because it is shoehorned into the existing knowledge work patterns and practices that focus on volume production. Thus, we also point to systemic innovation, saying that it must be performed to change these conventional practices before they are applied to support the technology. The two objects we have presented are meant to help enable systemic innovation.

Essentially, the two objects play the role of both new technology, and at a meta-level, as boundary objects. As technology, the VMO and the DMO each provide a set of affordances that support and enable new work patterns and practices. Thus, the two objects can be used to build good knowledge work ecologies as described in Chapter 3 and Chapter 4, respectively. As boundary objects, they act as a communication channel between two technical communities. The purpose of this is to enable further development of the objects, both theoretically as concepts and technically as software tools. In other words, our objects help different communities to communicate and collaborate in the development of new sociotechnical systems, which is what we argue is necessary for solving complex issues, such as information overload.

After having presented our two objects, we designed and implemented a simple prototype that was meant to show the main concepts of our two objects, as well as the interactions made possible by combining them. Then, we tested their capabilities as boundary objects by inviting

people from two different technical communities, tool makers and systemic innovators, to test our prototype and answer a questionnaire. We created one questionnaire with questions tailored for the tool maker evaluators and one for the systemic innovator evaluators. Finally, the answers were collected and the best feedback presented.

8.2 Critical Assessment

Initially, when we started to work with this thesis, it was only supposed to be about the Value Matrix Object and algorithms that could determine the value of an information resource. The idea was that by letting each resource having a value, glut could be eliminated, which would indirectly reduce information overload. Thus, time was spent reading what little already existed about the VMO, as well as creating a taxonomy of ways to collect data and understanding the information overload problem. However, we eventually decided that the VMO needed to exist in the same environment as the Domain Map Object to get the full power out of both objects. Therefore, this thesis was expanded to include both objects. Both of the objects were already described at a conceptual level in Karabeg and Lachica's article *Knowledge Federation as a Principle of Social Organization of Knowledge Creation and Sharing*. In this thesis, we have simply expanded on the concepts.

Further, we read about how other software approached the overload problem and thought about what our contribution would be. It became apparent that technology alone is probably not enough since information overload still exists, despite the powerful technology trying to handle it. After much thinking and discussion, we found that it must be the conventional practices of knowledge work, i.e. the sole focus on volume production, that were the root cause of information overload. Therefore, these practices and associated reward systems must be changed so that other activities of knowledge work, such as organization and evaluation, are also encouraged and rewarded. Thus, the VMO and DMO evolved from simple software tools to also be boundary objects that let different communities communicate and together create new sociotechnical systems.

In parallel with the development of our two object concepts we developed a prototype implementation of them. Since the two objects changed over time, so did the prototype. We even started with a prototype implementation while the Value Matrix Object was our sole focus. In other words, a lot of time was actually wasted during the prototype development. In hindsight, we should have waited until the theoretical foundations for the VMO and the DMO were completed in the context of this thesis before starting to work with an implementation.

For the final prototype iteration, we decided to start essentially from scratch. We went through a real design step using design patterns before starting with the implementation process, instead of developing the prototype ad hoc as we had done previously. Luckily, we were able to reuse parts of the code from the previous iteration, which obviously saved us some time.

The part of the thesis that we are the most unhappy about is the evaluation part. The reason for this is two-fold. First, it was rushed and planned only a couple of hours before the invitations to evaluate the prototype were sent out. From the results, it became apparent that more thought should have been put into the evaluation step, such as presenting an example of how the prototype can be used. It was daunting to see that feedback mostly consisted of the evaluators being confused. The other part we were unhappy about was the low turn out. We only received four answers for the questionnaires. Most likely, this was because the evaluation was conducted in the middle of the summer when most are on vacation.

Additionally, we could have conducted a usability test, observing people trying the prototype before sending out invitations to test the boundary object properties. Then, we could have remedied the usability problems that were complained about in the questionnaire responses and

perhaps received better results.

In hindsight, better results could have been achieved, and problems and redesigns avoided if we had made different decisions along the way. However, this is a part of the learning process: To make mistakes, take wrong decisions, and to learn from them. Additionally, what we experienced is also normal in a development cycle: Designing, implementing, testing, analyzing and then starting all over again, to remedy the problems that were found. Albeit, possibly in a more controlled fashion.

8.3 Further Work

8.3.1 Improving the Prototype

The first improvements to the prototype would be to address the issues that were pointed out during the evaluation. Specifically, GUI usability problems must be address to make the prototype more intuitive. The first step would be to make sure that functionality that is not useful in the current context is not shown. For example, in the prototype the sliders that are used to modify the resources that are shown are present even if we are not currently in the context where resources are shown. The same is true for the search field and the buttons that let us leaf through resources.

Furthermore, a help button or similar should be implemented that document the usage of the prototype, so that those who require additional information and explanation are able to get it without being frustrated. Additionally, the prototype should provide an explanation of the threshold sliders and how they work.

For even further development there are a lot of things that can be done. For example, more sliders could be defined and implemented. Additions include a ”popularity“ slider which can be measured criteria such as the number of visits a resource has. Further, more data collection methods for the existing sliders can be implemented. The schemes discussed in Chapter 4 would be a good start. With regard to completely new functionality, introducing sorts in addition to the sliders would be a good start. At the moment, knowledge resources are sorted on their QRI-score in descending order. Users have no way of changing how resources are sorted. However, this would be something they might want. For example, sorting on author, popularity or perhaps quality alone would be suitable sorts to start with.

Finally, we suggest that it would be appropriate to move gradually away from the Flash environment. While it is good for quickly developing high fidelity prototypes, it lacks when it comes to production versions. The biggest argument against Flash is that mobile apps become increasingly popular. To keep the software web based, Javascript, HTML and CSS could be used and still be accessible for most mobile platforms. Otherwise, a pure mobile application could be developed using Java or similar programming language.

Obviously, during further development different communities should be involved. For example, experts in reputation management should be involved in the development of the relevant algorithms. This way, we use the boundary object properties of our two building blocks.

8.3.2 Developing VMO and DMO Further

Naturally, there is also room for further development and improvement of the Value Matrix Object and Domain Map Object concepts. For example, a taxonomy of dimensions and evaluation schemes for both knowledge resource value matrices and user value matrices can be created. Doing this would greatly improve the options system developers have when integrating the VMO building block in software. Next, the reputation management system component could be fur-

ther explored. Especially for user value matrices. Finally, the interaction between knowledge resource value matrices and user value matrices should be investigated as they naturally impact each other. How can this fact help us with creating a knowledge work ecology?

For the Domain Map Object, there are at least two avenues that can be examined. The first is the various views that can be provided for resources. We have so far only explored a pure view that shows resources in relation to topics. Knowledge Cartography, which we presented in Chapter 2, is a possible way to go. Examples of other views can be to show specific areas where research is lacking or show knowledge resources relative to people instead of topics. That is, a connection between researchers and the knowledge they produce. The second avenue is to explore the possible relationship between elements, topics and resources, on a domain map. Inspiration for this can be found in Topic Maps, where such relationships exist. Indirectly, this would make it easier to create new views and maps over resources.

Bibliography

[1] Alfarez Abdul-Rahman and Stephen Hailes. "Supporting Trust in Virtual Communities". In: 2000, pp. 4–7.

[2] Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. 1st ed. New York, NY, USA: Cambridge University Press, 2008. ISBN: 0521880386, 9780521880381.

[3] Jackie Assa, Daniel Cohen-Or, and Tova Milo. "Displaying Data in Multidimensional Relevance Space with 2D Visualization Maps". In: *In IEEE Visualization '97*. IEEE Computer Society Press, 1997, pp. 127–134.

[4] J. Barzun, D. Diderot, and R. Bowen. *Rameau's Nephew and Other Works*. Library of liberal arts. Macmillan, 1964.

[5] Tim Berners-Lee, James Hendler, and Ora Lassila. "The Semantic Web". In: *Scientific American* 284.5 (May 2001), pp. 34–43. URL: `http://www.scientificamerican.com/article.cfm?id=the-semantic-web`.

[6] M. Biezunski, S. Newcomb, and S. Pepper. "ISO/IEC 13250:2002, Topic Maps". In: (2002).

[7] Michel Biezunski. *Topic Maps at a glance*. accessed September 20, 2011. URL: `http://www.infoloom.com/tmsample/bie0.htm`.

[8] Sonja Buchegger and Jean-Yves Le Boudec. *A Robust Reputation System for Mobile Ad-hoc Networks*. Tech. rep. Proceedings of P2PEcon, 2003.

[9] Sonja Buchegger and Jean-Yves Le Boudec. "The Effect of Rumor Spreading in Reputation Systems for Mobile Ad-Hoc Networks". In: *In Proceedings of WiOpt 03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, Sophia-Antipolis*. 2003.

[10] Tony Buzan and Barry Buzan. *The Mind Map Book*. 2nd ed. BBC Books, 1995.

[11] Stuart K. Card, J. D. Mackinlay, and Ben Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Academic Press, 1999.

[12] Matt Carmichael. *Edward Tufte: The AdAgeStat*. accessed June 1, 2012. URL: `http://adage.com/article/adagestat/edward-tufte-adagestat-q-a/230884/`.

[13] Rich Caruana and Alexandru Niculescu-Mizil. "An Empirical Comparison of Supervised Learning Algorithms". In: *In Proc. 23 rd Intl. Conf. Machine learning (ICML'06*. 2006, pp. 161–168.

[14] Alphonse Chapanis. "Words, words, words." In: *Human Factors*. 1965, pp. 1–17.

[15] J. Charpentier, M. Charpentier, and D. Diderot. *L'Encyclopédie*. Littérature Bordas. Les Éditions Bordas, 1967.

[16] Hsinchun Chen et al. "Internet Browsing and Searching: User Evaluations of Category Map and Concept Space Techniques." In: *JASIS* 49.7 (1998), pp. 582–603. URL: `http://dblp.uni-trier.de/db/journals/jasis/jasis49.html#ChenHSS98`.

[17] Mao Chen. "Computing and using reputations for internet ratings". In: *In Proceedings of the 3rd ACM Conference on Electronic Commerce*. ACM Press, 2001, pp. 154–162.

[18] Nick Collins. *Email has turned us into 'lab rats'*. accessed September 13,2011. Dec. 2010. URL: http://www.telegraph.co.uk/science/science-news/8184149/Email-has-turned-us-into-lab-rats.html.

[19] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine Learning* 20 (3 1995), pp. 273–297.

[20] Matt Cutts. *Google does not use the keyword meta tag in web ranking*. accessed February 14, 2012. Sept. 2009. URL: http://googlewebmastercentral.blogspot.no/2009/09/google-does-not-use-keywords-meta-tag.html.

[21] B. V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. Los Alamitos, CA: IEEE Computer Society Press, 1991.

[22] Martin Davies. "Concept mapping, mind mapping and argument mapping: what are the differences and do they matter?" In: *Higher Education* (Nov. 2010), pp. 1–23.

[23] Debategraph. *What is Debategraph?* accessed December 17,2011. URL: http://debategraph.org/home.

[24] N.K. Denzin and Y.S. Lincoln. *The Sage Handbook of Qualitative Research*. Sage Handbook Of... Sage, 2011. ISBN: 9781412974172. URL: http://books.google.no/books?id=qEiC-_ELYgIC.

[25] M. Dewey. *Dewey Decimal Classification, DDC 23*. Dewey Decimal Classification & Relative Index. OCLC Online Computer Library Center, 2011.

[26] Michael Douma, Ivan Angelov, and Sathish Menon. "Finding Information: Factors that Improve Online Experiences". In: *Journal of Interaction Recipes* 3 (2008).

[27] Michael Douma et al. "SpicyNodes: Radial Layout Authoring for the General Public." In: *IEEE Trans. Vis. Comput. Graph.* 15.6 (2009), pp. 1089–1096.

[28] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. 2. Wiley, 2001.

[29] Paul Farrand, Fearzana Hussain, and Enid Hennessy. "The efficacy of the 'mind map' study technique". In: *Medical Education* 36.5 (2002), pp. 426–431.

[30] Janet Ferguson. *Evaluating Web Information*. accessed August 15,2011. URL: http://bullpup.lib.unca.edu/library/lr/evalweb.html.

[31] Luciano Floridi. "Web 2.0 vs. the Semantic Web: A Philosophical Assessment". In: *Episteme* 6.1 (2009), pp. 25–37.

[32] Francis Galton. "Vox Populi". In: *Nature* 75 (1907), pp. 450–451.

[33] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. ISBN: 978-0-201-63361-0.

[34] Lars M. Garshol. "Metadata? Thesauri? Taxonomies? Topic maps! Making sense of it all". In: *Journal of Information Science* 30.4 (2004), pp. 378–391.

[35] Bryan Getting. *What Are Tags And What Is Tagging?* accessed February 14, 2012. Oct. 2007. URL: http://www.practicalecommerce.com/articles/589-What-Are-Tags-And-What-Is-Tagging-.

[36] C Gnoli. "Ten long-term research questions in knowledge organization". In: *Knowledge Organization* 35.2 (2008), pp. 137–149.

[37] Gary Grossman and Emmy Huang. *Actionscript 3.0 overview*. accessed January 13,2012. June 2006. URL: http://www.adobe.com/devnet/actionscript/articles/actionscript3_overview.html.

[38] Miniwatts Marketing Group. *Internet Users in the World*. accessed June 1,2012. URL: http://internetworldstats.com/stats.html.

[39] Miško Hevery. *Clean Code Talks - Global State and Singletons*. accessed February 28 ,2012. Nov. 2008. URL: http://googletesting.blogspot.com/2008/11/clean-code-talks-global-state-and.html.

[40] L. Hill et al. "Integration of knowledge organization systems into digital library architectures: Position paper". In: *Thirteenth ASIS&T SIG/CR Workshop on Reconceptualizing Classification Research* (2002).

[41] Birger Hjørland. "What is Knowledge Organization (KO)?" In: *Knowledge Organization* 35.2 (2008).

[42] Birger Hjørland and Karsten N. Pedersen. "A substantive theory of classification for information retrieval". In: *Journal of Documentation* 61.5 (May 2005), pp. 582–597.

[43] Gail Hodge. *Systems of Knowledge Organization for Digital Libraries: Beyond Traditional Authority Files*. Council on Library and Information Resources, Apr. 2000.

[44] J. Houghton. *Round holes not circular orifices.* 1968.

[45] IMDB. *The user votes average on film X is 9.4, so it should appear in your top 250 films listing, yet it doesn't. Why?* accessed December 13,2011. URL: http://www.imdb.com/help/search?domain=helpdesk_faq&index=1&file=notintop250.

[46] Neil Ingebrigtsen. *Understanding Information Overload*. accessed August 12,2011. URL: http://www.infogineering.net/understanding-information-overload.html.

[47] Ling Jiang et al. "The use of concept maps in educational ontology development for computer networks". In: *Granular Computing, 2008. GrC 2008. IEEE International Conference on*. 2008, pp. 346 –349.

[48] S. Jones. "Why can't leaflets be logical?" In: *New Society*. 1964, pp. 16–17.

[49] Audun Jøsang, Shane Hird, and Eric Faccer. "Simulating the Effect of Reputation Systems on e-Markets". In: *In Proc. of the 1st International Conference on Trust Management*. 2003.

[50] Audun Jøsang and Roslan Ismail. "The Beta Reputation System". In: *In Proceedings of the 15th Bled Electronic Commerce Conference*. 2002.

[51] Audun Jøsang, Roslan Ismail, and Colin Boyd. *A Survey of Trust and Reputation Systems for Online Service Provision*. 2006.

[52] Audun Jøsang and Walter Quattrociocchi. "Advanced Features in Bayesian Reputation Systems". In: *Proceedings of the 6th International Conference on Trust, Privacy and Security in Digital Business*. TrustBus '09. Springer-Verlag, 2009, pp. 105–114.

[53] Y. Kagolovsky and J. R. Moehr. "A new approach to the concept of "relevance" in information retrieval (IR)". In: *Medinfo 2001: Proceedings of the 10th World Congress on Medical Informatics*. 2001, pp. 348–352.

[54] Cem Kaner, Jack L. Falk, and Hung Quoc Nguyen. *Testing Computer Software, Second Edition*. 2nd. New York, NY, USA: John Wiley & Sons, Inc., 1999. ISBN: 0471358460.

[55] D. Karabeg and R. Lachica. “Knowledge Federation as a Principle of Social Organization of Knowledge Creation and Sharing”. In: (Oct. 2008).

[56] D. Karabeg, R. Lachica, and S. Rudan. “Quality, Relevance and Importance in Information Retrieval with Fuzzy Semantic Networks”. In: (2008).

[57] Dino Karabeg. “Informing Must Be Designed”. 2009.

[58] Dino Karabeg. “Knowledge Federation - An Enabler of Systemic Innovation”. In: (2011).

[59] Dino Karabeg and Benjamin Johansson. “Boundary Objects for Online Knowledge Management”. Note: Written as a supplement and introduction to this thesis.

[60] B.D. Klein. “User perceptions of data quality: Internet and traditional text sources”. In: *The Journal of Computer Information Systems* 41 (2001), pp. 9–18.

[61] S.A Knight and J.M. Burns. “Developing a Framework for Assessing Information Quality on the World Wide Web”. In: *Informing Science Journal* 11 (2005), pp. 159–172.

[62] T. Kohonen, M. R. Schroeder, and T. S. Huang, eds. *Self-Organizing Maps*. 3rd. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001.

[63] Maurice de Kunder. *The Size of the World Wide Web (The Internet)*. accessed June 1,2012. URL: http://www.worldwidewebsize.com/.

[64] Silex Labs. *amfPHP*. accessed January 13,2012. 2008. URL: http://www.silexlabs.org/amfphp/.

[65] Joe Lamantia. *Tag Clouds Evolve: Understanding Tag Clouds*. accessed February 14, 2012. URL: http://www.joelamantia.com/ideas/tag-clouds-evolve-understanding-tag-clouds.

[66] Valerie Landau, Eileen Clegg, and in conversation with Douglas. *The Engelbart Hypothesis: dialogs with Douglas Engelbart*. 2st Edition. NextPress, Nov. 2009.

[67] Last.fm. *Top Tags*. accessed February 14, 2012. URL: http://www.last.fm/charts/toptags.

[68] L. Laudan. *Progress and its problems: towards a theory of scientific growth*. Philosophy of science. University of California Press, 1978.

[69] Ting Liu et al. “An investigation of practical approximate nearest neighbor algorithms”. In: MIT Press, 2004, pp. 825–832.

[70] Peter Lyman and Hal R. Varian. “How Much Information?” In: (2003). URL: http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/printable_report.pdf.

[71] G. Mahoney. “Trust, Distributed Systems, and the Sybil Attack”. In: PANDA Seminar Talk, University of Victoria, Canada. 2002.

[72] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[73] Marshall McLuhan. *The Best of Ideas. CBC Radio*. 1967.

[74] Knowledge Federation Members. *Knowledge Federation*. accessed April 10, 2011. URL: http://knowledgefederation.ning.com/.

[75] Evan Miller. *How Not To Sort By Average Rating*. accessed September 8, 2011. URL: http://www.evanmiller.org/how-not-to-sort-by-average-rating.html.

[76] Nolan Miller, Paul Resnick, and Richard Zeckhauser. *Eliciting Honest Feedback in Electronic Markets*. Working Paper Series rwp02-039. Harvard University, John F. Kennedy School of Government, Sept. 2002. URL: http://ideas.repec.org/p/ecl/harjfk/rwp02-039.html.

[77] R.G. Miller. *Simultaneous statistical inference*. Springer series in statistics. Springer-Verlag, 1981.

[78] E. Mumford. *Sociotechnical Systems Design: Evolving Theory and Practice*. Working paper series (Manchester Business School). Manchester Business School, 1985.

[79] F.W. Nietzsche, W.A. Kaufmann, and R.J. Hollingdale. *The Will To Power*. A Vintage Giant. Vintage Books, 1968.

[80] Joseph D. Novak and Alberto J. Canas. *The Theory Underlying Concept Maps and How to Construct and Use Them*. research report 2006-01 Rev 2008-01. Florida Institute for Human and Machine Cognition, 2006. URL: http://cmap.ihmc.us/Publications/ResearchPapers/TheoryCmaps/TheoryUnderlyingConceptMaps.htm.

[81] OCLC. *Dewey Services*. accessed June 11, 2012. URL: http://www.oclc.org/dewey/.

[82] Lawrence Page et al. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report 1999-66. Stanford InfoLab, 1999. URL: http://ilpubs.stanford.edu:8090/422/.

[83] W.A. Pasmore and J.J. Sherwood. *Sociotechnical systems: a sourcebook*. University Associates, 1978.

[84] Marco Quaggiotto. "Knowledge Atlas: a cartographic approach to the social structures of knowledge". In: (2008).

[85] Marco Quaggiotto. *This is Knowledge Cartography*. accessed December 7,2011. URL: http://knowledgecartography.org/.

[86] S.R. Ranganathan. *Colon Classification*. Madras Library Association, 1933.

[87] S.R. Ranganathan. *Philosophy of library classification*. Library research monographs. E. Munksgaard, 1951.

[88] Kate Ray. *Web 3.0*. accessed June 4, 2012. URL: http://vimeo.com/11529540.

[89] P. Resnick and R. Zeckhauser. "Trust Among Strangers in Internet Transactions: Empirical Analysis of Ebay's Reputation System". In: *The Economics of the Internet and E-Commerce* 11 (2002).

[90] P. Resnick et al. "Reputation Systems". In: *Communications of the ACM* (Dec. 2000).

[91] P. Resnicka et al. "The Value of Reputation on Ebay: A Controlled Experiment". In: *Experimental Economics* 9 (2006), pp. 79–101.

[92] Y. Rogers, H. Sharp, and J. Preece. *Interaction Design: Beyond Human-Computer Interaction*. John Wiley and Sons Ltd, 2002.

[93] Andreas Schlosser, Marco Voss, and Lars Brückner. "On the Simulation of Global Reputation Systems". In: *Journal of Artificial Societies and Social Simulation* 9.1 (2005), p. 4. ISSN: 1460-7425. URL: http://jasss.soc.surrey.ac.uk/9/1/4.html.

[94] Chrysanthos Dellarocas Sloan. *Immunizing Online Reputation Reporting Systems Against Unfair Ratings and Discriminatory Behavior*. 2000.

[95] Jonathan B. Spira. *Interface - Information Overload: None Are Immune*. accessed September 23 ,2011. Sept. 2011. URL: http://www.information-management.com/issues/21_5/information-overload-none-are-immune-10021096-1.html.

[96] StackOverflow. *How does "Reputation" work*. accessed January 16,2012. 2011. URL: http://meta.stackoverflow.com/questions/7237/how-does-reputation-work.

[97] Google Staff. *FAQ: Crawling, indexing and ranking*. accessed September 10, 2011. URL: https://sites.google.com/site/webmasterhelpforum/en/faq--crawling--indexing---ranking#pagerank.

[98] PhpMyAdmin Staff. *About*. accessed January 13,2012. URL: http://www.phpmyadmin.net/home_page/index.php.

[99] The Straigt Dope Staff. *What's so great about the Dewey Decimal System?* accessed June 11, 2012. URL: http://www.straightdope.com/columns/read/2238/whats-so-great-about-the-dewey-decimal-system.

[100] Susan L. Star and James R. Griesemer. "Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39". In: *Social Studies of Science* 19.3 (1989), pp. 387–420.

[101] D.M. Strong and R.Y. Wang. "Beyond accuracy: What data quality means to data consumers". In: *Journal of Management Information Systems* (1996), pp. 5–33.

[102] D.M. Strong, R.Y. Wang, and Y.W. Lee. "Data Quality in context". In: *Communications of the ACM* 40 (1997), pp. 103–110.

[103] Danny Sullivan. *Meta Keywords Tag 101: How To Legally Hide Words On Your Pages For Search Engines*. accessed February 14, 2012. Sept. 2007. URL: http://searchengineland.com/meta-keywords-tag-101-how-to-legally-hide-words-on-your-pages-for-search-engines-12099.

[104] James Surowiecki. *The Wisdom of the Crowds*. Doubleday, Anchor, 2004.

[105] Eushiuan Tran. *Verification/Validation/Certification*. Tech. rep. Carnegie Mellon University, 1999.

[106] Edward Tufte. *Envisioning information*. Graphics Press, 1990.

[107] Josh Tynjala. *Runtime Enforcement of Abstract Classes in AS3*. accessed January 13,2012. Aug. 2007. URL: http://joshblog.net/2007/08/19/enforcing-abstract-classes-at-runtime-in-actionscript-3/.

[108] Peter Van Dijck. *Introduction to XFML*. accessed June 2, 2012. URL: http://www.xml.com/pub/a/2003/01/22/xfml.html.

[109] Mark Warschauer and Deborah Healey. "Computers and language learning: an overview". In: *Language Teaching* 31.02 (1998), pp. 57–71.

[110] Edwin B. Wilson. "Probable Inference, the Law of Succession, and Statistical Inference". In: *Journal of the American Statistical Association* 22.158 (1927), pp. 209–212.

[111] Christopher C. Yang, Hsinchun Chen, and K. K. Hong. "Visualization tools for self-organizing maps". In: *Proceedings of the fourth ACM conference on Digital libraries*. DL '99. Berkeley, California, United States: ACM, 1999, pp. 258–259.

[112] Christopher C. Yang, Hsinchun Chen, and Kay Hong. "Visualization of large category map for internet browsing". In: *Decis. Support Syst.* 35.1 (Apr. 2003), pp. 89–102.

[113] Bin Yu and Munindar P. Singh. "Detecting deception in reputation management". In: *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM Press, 2003, pp. 73–80.

[114] Marcia Lei Zeng. "Knowledge Organization Systems (KOS)". In: *Knowledge Organization* 35.2 (2008).

[115] J. Schneider et al. “Disseminating Trust Information in Wearable Communities”. In: Proceddings of the 2nd International Symposium on Handheld and Ubiquitous Computing. Sept. 2000.

Appendices

Appendix A

Client Side Source Code

This appendix will go through the source code classes for the client side part of the prototype. Additionally, we will describe how new versions of the prototype can be built. All of the files we talk about in this appendix can be found in the following SVN repository: *https://subversion.assembla.com/svn/vmo-dmo-clientside/*

A.1 The Classes

This is a list of the files that, when compiled, creates a build of the prototype:

- **AbstractGUIControlPanelView**. Is an abstract class that contains common control panel functionality and is subclassed by more specific control panel classes.
- **AbstractGUISideView**. This abstract class contains common functionality for the side view panels and is subclassed by more specific control panel classes.
- **AddResourceCommentWindow**. Contains functionality for submitting knowledge resource comments through a pop up window.
- **AddResourceWindow**. Contains functionality for submitting and adding new knowledge resources to topics on the domain map.
- **AddTopicWindow**. Contains functionality for submitting and adding new topics to the domain map.
- **ArrayIterator**. Is the concrete implementation of an iterator. This particular one can be used to iterate through arrays.
- **CommentCollection**. Is a collection that contains ResourceComment objects and functionality to manipulate the collection.
- **ComputationEngine**. This class is responsible for taking raw data from value matrices and compute various scores for resources that are later used to find the set of resources that matches a given query.
- **ControlPanelLoggedIn**. Is one of the states for the control panel. This state is active when users view knowledge resources, and provides functionality, including the threshold sliders, that let users make queries over them.

- **ControlPanelLoggedInUsers**. Is also a state for the control panel. It is active when users view user resources and provides the sliders that filter them.
- **ControlPanelLoggedOut**. The default state for the control panel. Lets users log in or register with the prototype.
- **DomainMapObject**. This class can be viewed as the main hub in the prototype. It creates the other main parts of the prototype and passes information between them. In addition, this class is responsible for drawing topics and resources.
- **DomainResource**. Contains information about a knowledge resource, such as its name, author, QRI-score and so on. Also creates an instance of a ResourceValueMatrixObject. The DomainMapObject draws instances of this class on the screen.
- **DomainTopicObject**. Very simple class that contains that name and id of a topic. The DomainMapObject draws instances of this class on the screen as topics that can be clicked on to unveil knowledge resources.
- **DropDownMenu**. Third party class that enables us to create drop down menus or lists. We use this class to create the menu where users select the topics a resource can belong to.
- **GUIViewStateMachine**. Is responsible for changing between the various GUI view states. For example from ControlPanelLoggedOut to ControlPanelLoggedIn.
- **Linkbox**. Third party class that contains the functionality related to the elements of the DropDownMenu. This class was slightly modified to suit our needs.
- **Main**. The first class that is executed. Instantiates a DomainMapObject.
- **MD5**. Contains the algorithm that MD5-encrypts passwords. This is a third party class that was downloaded so we did not have to implement this common algorithm ourselves.
- **PlainButton**. Very simple class that contains the functionality, such as being able to click on it, for a button graphical element.
- **QueryEngine**. Is responsible for finding the set of resources that matches a query. Also keeps track of the thresholds users have set using the control panel sliders.
- **RegisterUserWindow**. Contains functionality for registering new user profiles through a pop up window.
- **ResourceCollection**. Is a collection that contains DomainResource objects and functionality to manipulate the collection.
- **ResourceComment**. Contains information about a comment provided from a user for a DomainResource, such as the actual comment itself and its author. The DomainMapObject draws instances of this class on the screen.
- **ResourceValueMatrix**. Contains raw data that the ComputationEngine class uses to compute reputation scores for DomainResource the ResourceValueMatrix is attached to.
- **Server**. This class is responsible for setting up a connection to the server, as well as process requests to and responses from the server.
- **SideViewLoggedOut**. Is the default state for the right hand side panel of the GUI.

- **SideViewProfile**. Also a part of the right hand side panel GUI. Shows information that is related to a user's profile.
- **SideViewResource**. Last class that is a part of the right hand side panel GUI. This class contains functionality that displays information about a knowledge resource, as well as exporting functionality that lets users rate a resource.
- **StrategyAverageRating**. Is a concrete implementation of a part of the strategy structure. The class is responsible for computing reputation scores, using the average of ratings algorithm, for knowledge resources. It is one of many potential interchangable computation strategies that implement the same interface.
- **SystemMessages**. This class is responsible for displaying various messages to users at appropriate times. For example, it tells the user that a knowledge resource has been added to the domain map or that an error has occured when trying to log in.
- **TopicCollection**. Is a collection that contains DomainTopic objects and functionality to manipulate the collection.
- **User**. Contains information about a user resource, such as its name, country of residence, place of work and so on. Also creates and stores an instance of a UserValueMatrixObject. The DomainMapObject draws instances of this class on the screen.
- **UserCollection**. Is a collection that contains User objects and functionality to manipulate the collection.
- **UserValueMatrixObject**. Contains raw data that the ComputationEngine class uses to compute author contribution and organization contribution scores for the User the UserValueMatrix is attached to.

A.2 Interfaces

This is a list of the current interfaces that various classes must implement. They are all contained within the *interface* folder in the same SVN repository where the rest of the source code is.

- **ICollection**. All collections implement this interface. Contains the functions that all collections must implement.
- **IDomainMapObject**. The DomainMapObject class implements this interface.
- **IIterator**. Defines the functions that all concrete iterators must implement.
- **IReputationComputationStrategy**. Defines the functions for all concrete strategies must implement in order to be interchangable.
- **ISystemMessages**. Is the interface that the SystemMessages class implements.

A.3 Third Party Packages

In our implementation we used two third party packages that allowed us to easily implement some functionality that otherwise would have required many extra hours of programming. These packages are found in the same SVN repository as our other source code.

Log in credentials for dummy users	
Username	Password
marco@uio.no	tool123
marius@uio.no	tool456
dino@uio.no	sys123
jack@uio.no	sys456
erich@uio.no	sys789

Table A.1: Log in credentials for dummy users

- **Caurina**. The classes in this package folder contains code that enables tweenings and other transitions for Flash applications. Tweenings can be likened to simple animations. This package is required by the DropDownMenu class. More information about the Caurina package can be found here: *http://code.google.com/p/tweener/*
- **Com**. The classes in this package folder contains the slider functionality which we use in the prototype's control panel. Both the user interface for the sliders and events that are invoked, are defined in these classes. More information can be found here: http://evolve.reintroducing.com/2008/08/14/as3/as3-sliderui-v15/

A.4 Building the Source Code

Making a new build of the prototype requires that the computer that compiles the source code has the Flash developer environment installed. Our prototype was built using Flash CS5, so it is recommended that a new build is compiled using this or a higher version.

A new build can easily be compiled by opening up Software.fla, which can be found in the same repository as the source code files, in Flash CS5 or better. Then, a compile command can be executed.

Software.fla also contains the library which stores the prototype's graphical elements. Thus, Flash CS5 can also be used to add new graphical elements to the prototype, or make changes to the already existing ones.

A.5 Running the Prototype

Other than running the .swf file that is produced from compiling the source code directly, it can also be integrated into a web page for public access. We have done this with the current prototype build, and it is currently located at *http://benjamij.at.ifi.uio.no/*. We also created a couple of dummy user profiles that can be used to try the prototype without registering and creating a completely new profile. The log in credentials for these profiles can be seen in table A.1.

Appendix B

Server Side Files

This appendix will go through the files that make up the server side of the prototype. All of the files we talk about in this appendix can be found in the following SVN repository: *https://subversion.assembla.com/svn/vmo-dmo-serverside/*. We will also

B.1 The Files

This is a list of the files that the server side of the prototype concists of:

- **SoftwareXX.swf**. This file is our compiled prototype, i.e. the software itself. The two Xs denotes the identity of the build. We do this in order to avoid cache problems. In other words, all new builds are given a unique identification number.
- **index.php**. This file is the web page that we inject the above file into. It is important that we remember to update this file so that the correct build is embedded whenever we make a new build.
- **amfphp/**. In the thesis proper we talk briefly about amfPHP as a third party software that makes it easier for the client side and server side of the prototype to communicate. More information about amfPHP can be found here: *http://www.silexlabs.org/amfphp/*
- **phpMyAdmin/**. We also briefly mentioned phpMyAdmin as a tool that helps us manage the database structure and content by providing a graphical user interface. The database can be accessed with phpMyAdmin through the following url:
  http://benjamij.at.ifi.uio.no/phpMyAdmin/. Logging in to the database requires a username and a password, which is *benjamij* and *vmomasterthesis*, respectively. More information about the software can be found here:
  http://www.phpmyadmin.net/home_page/index.php.
- **VMOServer.php**. This file is located in the amfPHP folder structure. More specifically at amfphp/Amfphp/Services/. It is in this folder where services that amfPHP runs are placed. In other words, VMOServer.php is the file the client, in our case Server.as, calls functions on in order to store data in or retrieve data from the database. VMOServer.php contains several functions that make database queries and return the result to the calling client.

Appendix C

Complete Evaluation Answers and Invitation Email

C.1 The Answers from the Questionnaires

Here we will present the answers we received from the two questionnaires. First, we list the answers recieved from the tool maker questionnaire before the answers from the systemic innovator questionnaire are presented. Each questionnaire has seven questions, which a total of four people answered. Three answered the tool maker questionnaire, while we only recieved one answer to the systemic innovator questionnaire. Please note that some evaluators left a couple of questions unanswered.

C.1.1 Answers from Tool Maker Questionnaire

1. When you first tried the prototype was it clear what you were supposed to do? If not, what was the issues?	
Evaluator number	Answer
1	Not really. For example, on the landing page, the View Knowledge button does nothing. On every other page, it takes me back to the landing page. Clicking on any of the "knowledge" buttons takes me to a page with just that button. One time only, clicking on Issues got me a circular display of some issues. That never happened again. I think the site crashed.
2	Sorry, but the tool is not intuitive. I have no clue, what I should do when I'm logged in. I tried the topic "tmra" but nothing happened.
3	It was not clear since the "gui" connected only loosely with the patterns usually encountered at websites

Table C.1: Tool maker questionnaire - answers to question 1

2. While using the prototype, did you feel overwhelmed by the amount of information provided on the screen? If yes, what would you change?	
Evaluator number	Answer
1	no, but I am left wondering what the site will look like when it has more than just those few categories. "Overwhelmed" is not the term I would use. "Confused" is closer. Why does the UI show me sliders when there is nothing to slide? I always thought that good UX meant not showing controls until they are meaningful. Showing them on the landing page implies they are ready to do something useful, but they are not useful until you are inside some Topic (if those are topics, what are all the other things you see when you click on one of them?) To me, they are more like categories, facets.
2	I did not feel overwhelmed by the amount of information. In contrast, I did not see any interesting information. I did only see some sliders etc, but what is the content and how this content can be used.
3	No

Table C.2: Tool maker questionnaire - answers to question 2

3. Currently, users are met with a set of topics they can click on in order to see the knowledge resources that are relevant to that topic. Further, they can click on a resource to view more information about it and evaluate it. Is this layout intuitive and understandable, or should it change? If yes, do you have any suggestions on what should be improved?	
Evaluator number	Answer
1	It's intuitive, but, for the most part, it didn't work that way. While the topics would show the link cursor, they did nothing when you click on them, except for the first pass. I closed the page and returned. It started working better, but I noticed that the resource I added to Tools was not showing.
2	
3	the prototype was not responsive in the way the question would let assume; aka I didn't see any see knowledge resources that are relevant to the topic

Table C.3: Tool maker questionnaire - answers to question 3

4. Are the sliders that modify what resources are shown understandable and intuitive? What can be done to improve how users filter the resources?	
Evaluator number	Answer
1	Nope. I have no clue what I am filtering on. Clicking the left arrow got me a dialog that said "The best resources are already displayed..." I had no clue I was choosing "best" anything. The only possible way to improve that would be a mouse-over which explains what each slider does. The explanation given in the pdf paper should be sufficient.
2	The sliders are intuitive. I like this idea. But I was a bit curious what happens when I move thee knowledge slider. Where is this "knowledge value" from? Why something disappears? This is not clear to the user -. and so I don't have any trust in this slider.
3	The sliders are really perfect / I love them ;) good work

Table C.4: Tool maker questionnaire - answers to question 4

5. From the viewpoint of a toolmaker, does the prototype provide all the functionality needed for the intended usage? If not, what is lacking?	
Evaluator number	Answer
1	What, precisely, is the intended usage? Keep in mind that I did not read the paper yet; I took the view of a naive user trying out a new knowledge mapping tool. I had many expectations from prior knowledge. Reading the paper didn't change much except for the explanation of the sliders, which still remain confusing. Who decides what the expertise level is? The person who creates the resource. How am I supposed to trust that value? How do I know the system isn't being gamed?
2	USABILITY and DOCUMENTAION For an average user like me it is not possible to grasp the idea behind the tool. The idea - which seems to be interesting - is hidden behind a user interface with very akward usability and no user documentation. Hence it is unfortunately not usable for productive usage.
3	as said ... some of its funktionallity is rather unintuitive

Table C.5: Tool maker questionnaire - answers to question 5

6. From the viewpoint of a toolmaker, if you could change anything about the prototype, what would it be?	
Evaluator number	Answer
1	That's a huge question. There is room for a zillion answers, or none based on the notion that it is already simple and somewhat elegant but leaves one to wonder what it will be like when it is dealing with real world resources rather than toys. My own experiment to add a resource appears to have failed; thus I am not able to see how it would behave when I add, say, 100 tools, all with the same expertise level. On thing: I'd be strongly motivated to move away from flash given that the world is going mobile.
2	The same: USABILITY and DOCUMENTATION Sorry for being that imprecise in ideas. But to have more concrete input it would be necessary for me to understand the tool - which is not possible at the moment.
3	

Table C.6: Tool maker questionnaire - answers to question 6

7. Other comments or suggestions?	
Evaluator number	Answer
1	It's a really good start. Keep in mind that my own set of expectations. I looked for connections among topics. Didn't see any. Didn't see an opportunity to create them. I clicked Add a Topic; it opened a pull down; I selected "Patterns". End of story. Nothing else to do No response, no action, no new topic. Not until I clicked Submit did it respond.The right hand pane where you can view a resource and add ratings is an interesting idea, but not intuitive. There are several actions possible, e.g. new topic and submit buttons are tied to each other, whereas the first impression is that submit has something to do with the whole pane. New Topic is ambiguous: what it is really doing is connecting, e.g. Key Point Dialog (should that be Dialogue) with the not-so-new category Patterns (my choice). That worked. Search is absolutely ambiguous. The first thing one might want to do is put in some term and click search. All one gets on the landing page is "Cannot search outside a topic" Huh?
2	Make the tool more simple and shiny.
3	

Table C.7: Tool maker questionnaire - answers to question 7

C.1.2 Answers from Systemic Innovator Questionnaire

1. Currently, there is a set of sliders that modify what resources are shown on the screen. Can you think of any other affordances that could do the job better?	
Evaluator number	Answer
1	I am underwhelmed by "quality". In what context is quality measured? By whom? Remember that what is high quality to one user might be garbage to another. Ditto for "importance"

Table C.8: Systemic innovator questionnaire - answers to question 1

2. Can you think of any relevant criteria other than time, knowledge, quality and importance that should have its own slider?	
Evaluator number	Answer
1	Frequently, one looks for things like "number of connections", "popularity" as measured by any number of criteria, one of which is the relative number of visits.

Table C.9: Systemic innovator questionnaire - answers to question 2

3. At the moment, users are rewarded within two categories. Either by contributing to the organization of existing knowledge by evaluation or by publishing new material. Can you think of other ways to reward users so that they feel inclined to use this prototype or a similar tool?	
Evaluator number	Answer
1	How, precisely, are users rewarded? I don't recall any reward other than a dialog opening to thank me, which I was forced to click to get rid of. Game mechanics, as Dino well knows, appears to be the way forward in reward mechanisms.

Table C.10: Systemic innovator questionnaire - answers to question 3

4. In the prototype, a knowledge resource's value in a context is computed using quality, importance and relevance dimensions. These dimensions are given by users in the form of ratings which a reputation engine collects and computes a score for. Can you think of other dimensions that could be relevant to the computation of a knowledge resource's value?	
Evaluator number	Answer
1	What, precisely, is the context in which I create that score? This question is almost the same as 2 above.

Table C.11: Systemic innovator questionnaire - answers to question 4

5. Can you think of other ways to evaluate the quality, importance and relevance dimensions other than through user ratings?	
Evaluator number	Answer
1	This is complex. Just a simple resource, itself, is meaningless, somewhat like the sound of a tree falling in the forest if nobody hears it. But, what does that resource do by being there? It supports epistemic communities by being a resource; just how it gets used becomes part of the quality evaluation, and that's liable to be different for different users. That a user clicked it at all indicates at the very least enough relevance that it raised curiosity. There is no way to track what that curiosity led to unless one puts a form in which allows the user to say how relevant it was *to that user at that time for that purpose*. This is the pragmatic inquiry, and the precise nature of "user", "temporal context", and "purpose" is going to be different for most all users, which makes gathering that information far more complex than can be captured in just a few numbers. There is a tendency to think in terms of a recursive waltz: first, you figure out what *topics* are in play (context definition), then you look at the relations among those topics, and then you look at the relative importance of those relations, etc, all the way down to the first turtle. This one's hard.

Table C.12: Systemic innovator questionnaire - answers to question 5

6. As a systemic innovator, is the functionality of the prototype sufficient in order to build systems that create an ecology of knowledge work? If not, what do you think is missing?	
Evaluator number	Answer
1	Not yet. It's a really valuable first step; it's the first operational prototype in my experience that tackles the enormously complex issue of acquiring information necessary to assess the value proposition for any and every resource in an epistemic ecosystem. What I think is missing would be the topic of several conference papers, but I've tossed a few first thoughts in questions above.

Table C.13: Systemic innovator questionnaire - answers to question 6

7. Other comments or suggestions?	
Evaluator number	Answer
1	Googling "systemic innovator" landed many hits, among them: http://www.socialinnovator.info/process-social-innovation/systemic-change/strategic-moves-accelerate-systems-change/designing-and-trialling-platforms-trigger-sy which attracted me there with a nice title. But, the attraction was based on an expectation that this might be a conference paper, or maybe a blog post. It is neither. Instead, it is a subject in what appears to be a really weak, not very intuitive topic map. At that point, my own ratings (using your tools) of that page would not be very high. All that, based on expectations and expectation failure. This is the same issue for every user of any epistemic portal. It stands in the way of gaining full access to the "true value" (whatever that means) of any resource. To tease out of every user greater precision entails a lot of work, an ontology that provides term stability (and which constrains the inquiry, possibly leaving out important concepts), and may end up discouraging serious participation. After all, the more things you ask of users, the lower the probability that users will do them. Still, we need to start somewhere. This is an important start.

Table C.14: Systemic innovator questionnaire - answers to question 7

C.2 The Content of the Invitation Email

For the evaluation part of this thesis we sent an email to a select group of people, inviting them to participate in the evaluation by testing the prototype and answering a questionnaire. In addition to the invitation, the purpose of the email was to provide some information about the thesis project and providing links to the prototype and the questionnaires. The following is one of those emails in full. The first part was written by this thesis' supervisor, Dino Karabeg, while the second part was written by the master student and writer of this thesis, Benjamin Johansson.

> Jack,
>
> By inviting you to take part in this experiment, we would like to:
>
> * share with you this particular way of working, which by creating this prototype we are only beginning to develop
>
> * invite you to contribute to it by sharing with us your impressions and related ideas and experience
>
> We have made an effort to design this experiment in such a way that it should be possible to complete it within 30 minutes.
>
> We invite you to answer the two questionnaires linked at the very bottom, in the light of the information provided below (we invite you to appear in two roles, as 'system designer' and as 'tool maker').
>
> I would like to add the following to Benjamin's rather thorough explanation, which is provided below:

The Domain Map Object (DMO) and the Value Matrix Object (VMO) are 'boundary objects' their key role is to enable communication between two technical communities, called here 'tool makers' and 'systemic innovators.' Systemic innovators belong to a multidisciplinary team whose task is to create innovative real-life socio-tehnical systemic solutions for, say, education, academic research or public informing, as enabled by new information technology. Examples of 'tool makers' are Topic Maps and Semantic Web experts. The boundary objects enable the systemic innovators to use 'hide' implementatin (technology) and 'export' functions, as they might suit the intended applications.

The poll we are inviting you to take part in is itself part of this prototype. Its role is to enable both target communities to take part in this designhence enable our objects to truly serve as a 'boundar' between those communities, i.e. as a communication channel.

We greatly appreciate your help, and hope you will find this experiment and the ideas it brings to life interesting and rewarding.

Warm regards,

Dino

Begin forwarded message:

From: Benjamin Johansson <benjamij@ifi.uio.no>
Date: June 27, 2012 4:12:30 PM GMT+02:00
To: Dino Karabeg <dino@ifi.uio.no>
Subject: surveys and info

This is my 1/2 page intro:

We send you this email to ask you to help evaluate a software prototype that M.Sc.student Benjamin Johansson has developed for his thesis: Value Matrix and Domain Map - Boundary Objects for Systemic Innovation. The evaluation consists of a questionaire containing 6 questions pluss an open field where you can write any suggestions or encouragements not covered by the initial questions.

Some background information about the project: The thesis revolves around the information overload problem, which is an ongoing problem in society. We claim that in order to solve this, and other, complex problems we are facing, we need to design technology and social systems that use it together instead of developing new technology that supports old work practices.

Thus, we introduce two objects which serve as building blocks to support our systemic innovation; The Domain Map Object and the Value Matrix Object. A Domain Map Object (DMO) is a knowledge management object that is assigned to a community of interest. It is an online representation of the community's domain of interest and provides the means to access and organize knowledge resources in a subject-centric way, while it aims to minimize both overload and visual load. In addition, a domain map offers the possibility of having multiple views of the domain it represents, for example showing various levels of detail or highlighting areas where knowledge is lacking. Members in a community may use the DMO to publish their papers, organize existing information in domain or locate information.

The Value Matrix Object (VMO) is an object that is associated with a resource in the domain represented by a DMO. A resource can either be a document

containing information or a person that is a member of the community. Thus, the VMO plays to similar, yet distinct roles. When attached to a resource, the VMO accumulates all data that could be relevant for computing the value of that resource given a query. This data is accumulated through, for example, ratings provided by members of the community about the quality or importance of the resource. The other role is when attached to a person. Then the VMO collects all information about the member's contribution, which is done so that undervalued contributions like organization and evalution of knowledge is suitably rewarded.

Our idea is that by combining the DMO and the VMO we can create a good knowledge-work ecology. We do this by not only rewarding the publishing of papers in a coherent system, but also by making it suitable for knowledge workers to spend their time organizing and evaluating already existing knowledge. Additionally, the VMO attached to knowledge resources enables the possibility that truly valuable resources are brought forward and highlighted, while the mediocre resources drown. Furthermore, this makes it possible for the valuable resources' authors to be suitably rewarded for their outstanding contribution.

Naturally, the prototype is just that, a prototype. It is designed to show the core concepts and its functionality and design are not final. More than likely, most of it will change in the future. Questions can be directed to Benjamin through benjamij@ifi.uio.no

More information can be found in the following article:
http://benjamij.at.ifi.uio.no/files/BoundaryObjects.pdf

Prototype link: benjamij.at.ifi.uio.no/

A couple of login credentials for "dummy" users. These are real people, but created by me. I created a couple of alternatives. It does not really matter who they log in to. I will manipulate the data for each of these users to that they are different. Currently they are "new" with no contribution or anything.

Toolmakers:
username: marco@uio.no
pw: tool123
username: marius@uio.no
pw: tool456
Systemic Innovators:
username: dino@uio.no
pw: sys123
username: jack@uio.no
pw:sys456
username: erich@uio.no
pw:sys789
Finally here are the two questionnaires:
Questionaire link toolmaker:
http://www.surveymonkey.com/s/VWWLV39
Questionaire link systemic innovator:
http://www.surveymonkey.com/s/VS3NCK7